Dongbu Robot

DR Visual Logic

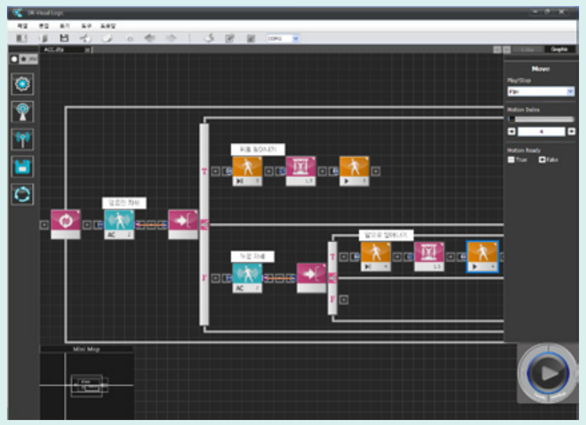
for **HOVIS Lite/Eco(plus)**

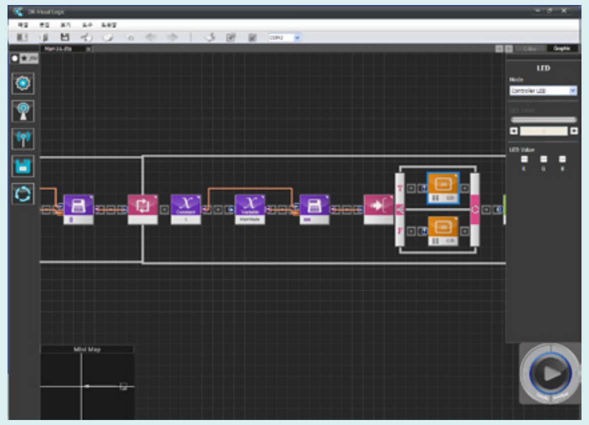
www.dongburobot.com

DR Visual Logic

For Hovis Lite, Eco Plus

CONTENTS





01 Installation

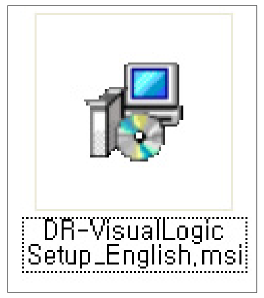
DR-Visual Logic Introduction

DR-Visual Logic is a Drag & Drop type graphic robot programming tool derived from the robot programming language developed by Dongbu Robot. DR-Visual Logic has been customized to work with Dongbu Robot DRC controller by modularizing the DRC functions. Drag & drop method using the mouse makes DR-Visual Logic easy to program even by the novices and by using the provided C-like tab, text codes converted from the graphic program can be viewed immediately. As the codes are similar to the C language, it will also help the novice programmers in learning the C language. DR-Visual Logic is one of the easiest and yet powerful programming tools in the market and its versatility makes it equally popular with novice and advanced users alike. Planned upgrade to the program to make it even more versatile and powerful includes upgraded DRC function modules, motion modules, and integrated simulation.

System Requirements

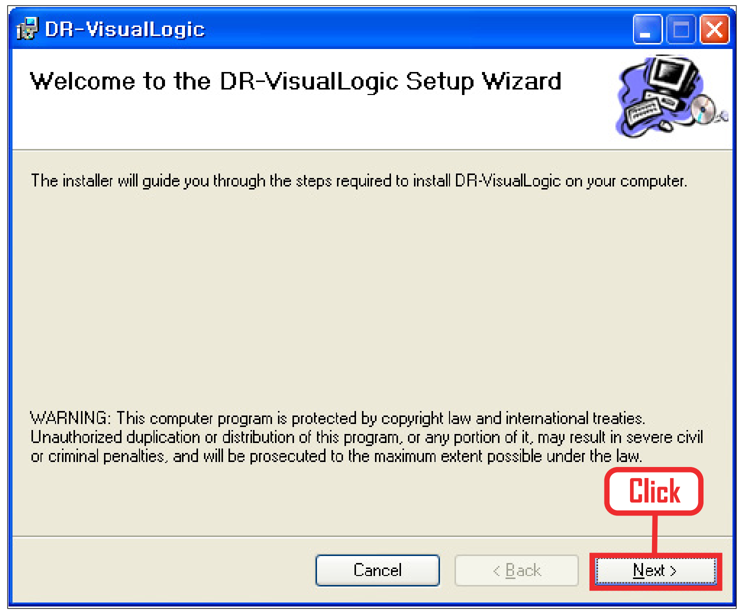
- Minimum Intel Pentium 800 Mhz
- Windows XP / Vista / 7
- Minimum 256 MB RAM
- Minimum 20 MB Hard disk space
- USB Port

Step by Step / From installation to test



01 Installation File

Click on the installation file.


DR-VisualLogic

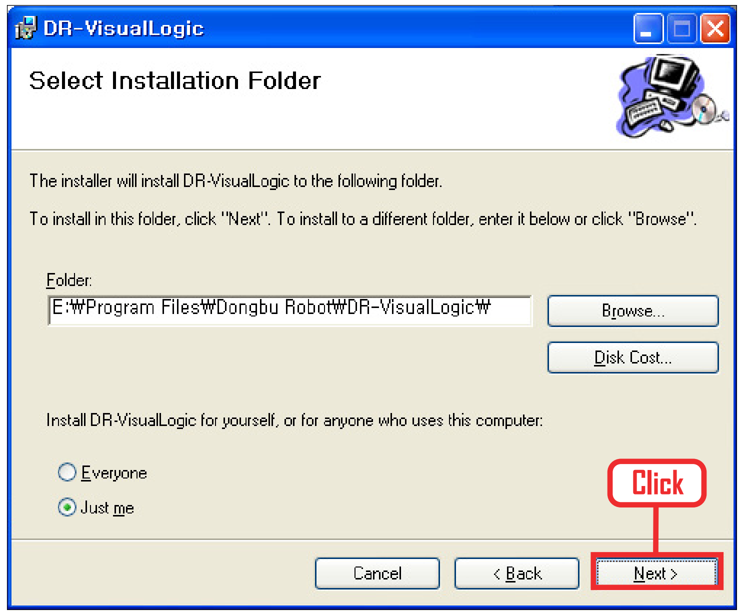
Welcome to the DR-VisualLogic Setup Wizard

The installer will guide you through the steps required to install DR-VisualLogic on your computer.

WARNING: This computer program is protected by copyright law and international treaties. Unauthorized duplication or distribution of this program, or any portion of it, may result in severe civil or criminal penalties, and will be prosecuted to the maximum extent possible under the law.

02 Start installation Wizard

Click "Next" button.


DR-VisualLogic

Select Installation Folder

The installer will install DR-VisualLogic to the following folder.

To install in this folder, click "Next". To install to a different folder, enter it below or click "Browse".

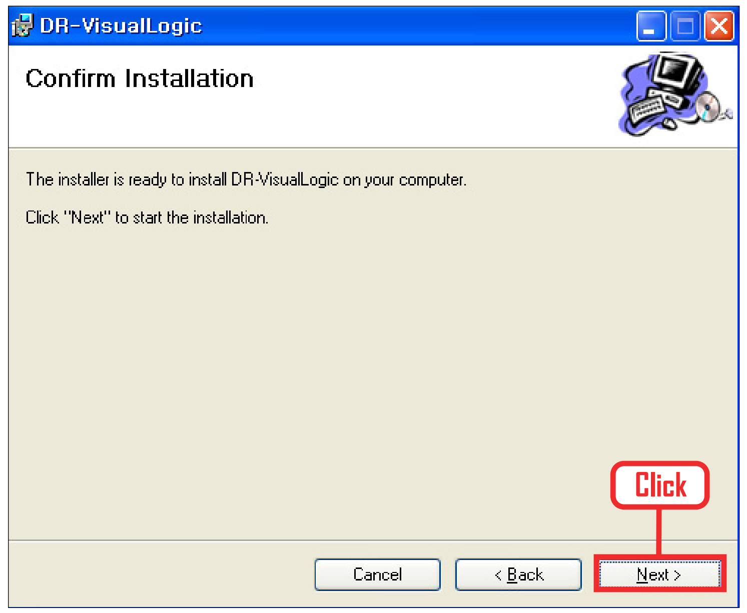
Folder: E:₩Program Files₩Dongbu Robot₩DR-VisualLogic₩

Install DR-VisualLogic for yourself, or for anyone who uses this computer:

Everyone [✓ Just me]

03 Select Installation Folder

Click "Next" button.


DR-VisualLogic

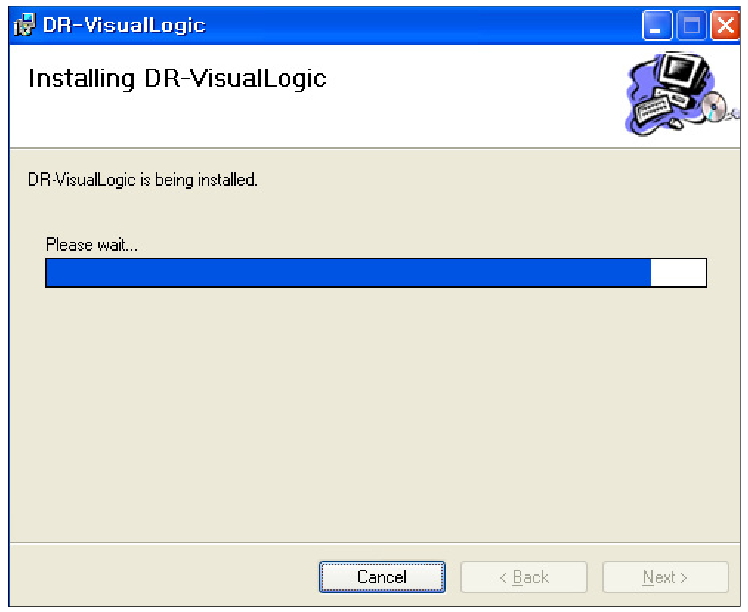
Confirm Installation

The installer is ready to install DR-VisualLogic on your computer.

Click "Next" to start the installation.

04 Confirm Installation

Click “Next” button.


DR-VisualLogic

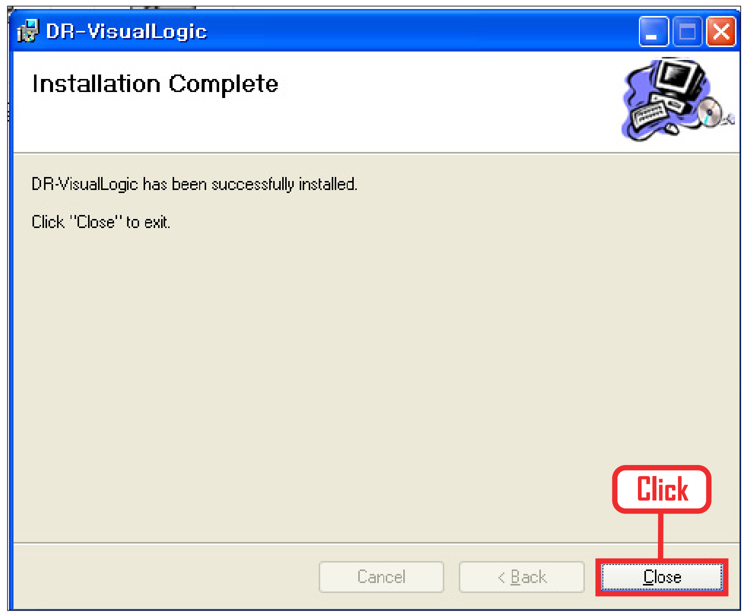
Installing DR-VisualLogic

DR-VisualLogic is being installed.

Please wait...

05 Start Installation

Starting installation.
Wait for the progress bar to end.


DR-VisualLogic

Installation Complete

DR-VisualLogic has been successfully installed.

Click "Close" to exit.

06 Finish Installation

Click “Close” button
Program installation complete.

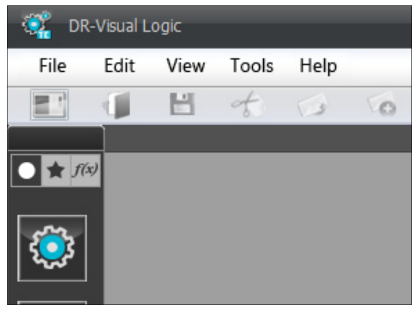
07 Check executable file

Check for the executable file, desktop shortcut icon and from Windows Start > All Programs > Dongbu Robot > DR-VisualLogic.
Click on the executable file to run the program.

Hello DR-Visual Logic 02

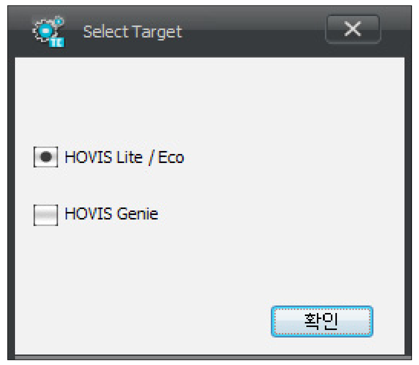
First Program Step by Step

16 Axis humanoid robot will have both its arms stretched out to the side when all the servo motors installed in the robot are aligned to the center. This example program will lower one arm to the side of the body.



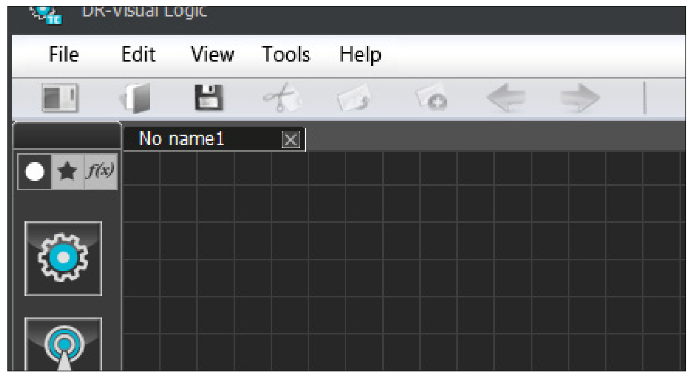
01 Create New File

Click on the left most icon in the tools to create new dts file.



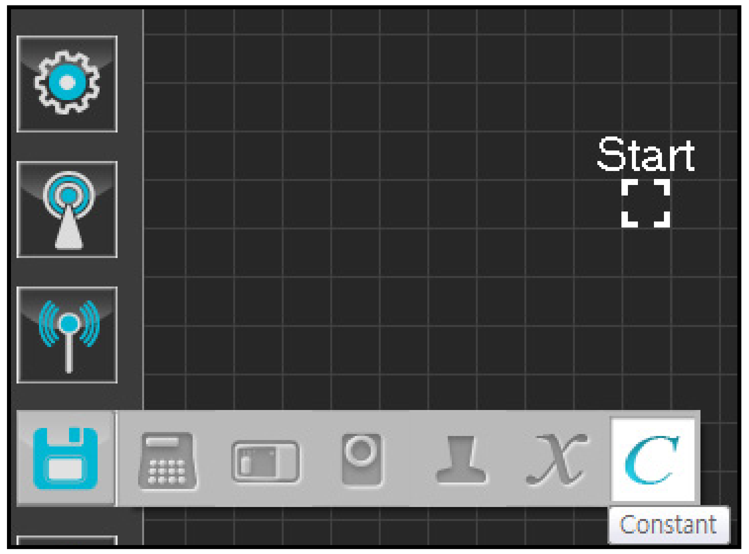
02 Select Robot

Newly created dts file requires user to select the robot to apply the new program. Select HOVIS Lite/Eco.



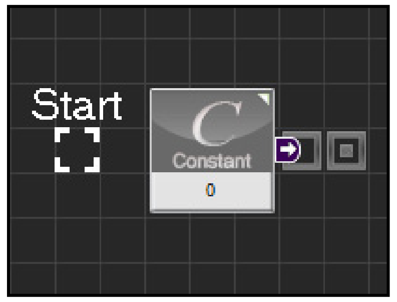
03 New Window

New file created.



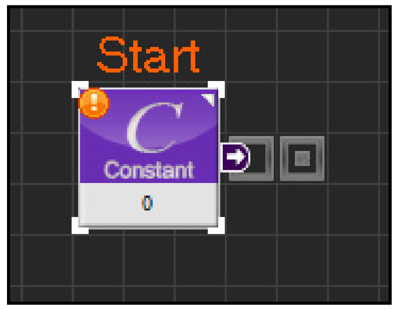
04 Select Module

Prior to placing the module, click to select the module from the module bar located on the left side.
Click Data > Constant module.



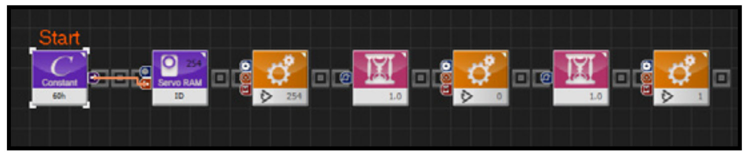
05 Place Module

When clicked, selected module will move following the mouse cursor. Another mouse click will place the module at the current cursor location. Drag and place the module at the Start Point.



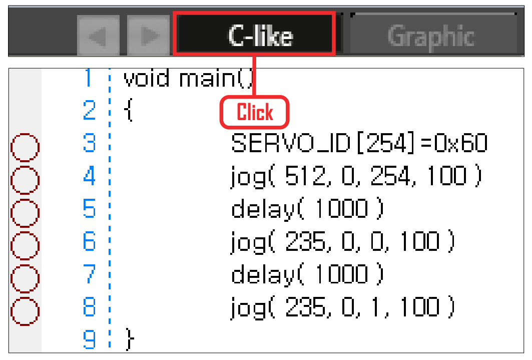
06 Start Programming

When module becomes docked to the Start Point properly, module becomes active and module image changes to color to show an active module as shown in the diagram to the left.



07 Entire Program

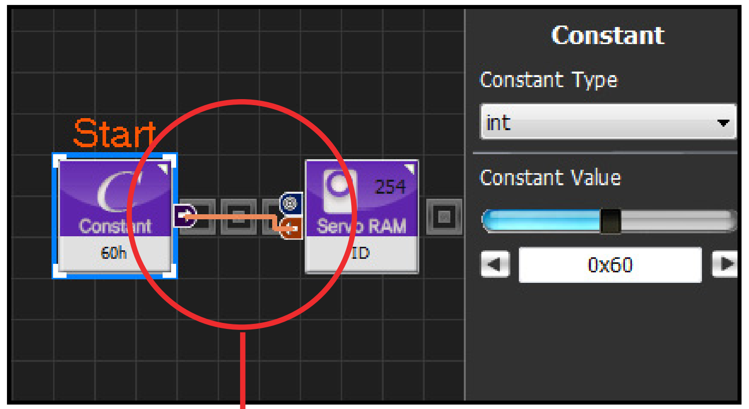
Diagram to the left shows the entire program that lowers the robot arm by operating the servo motor.



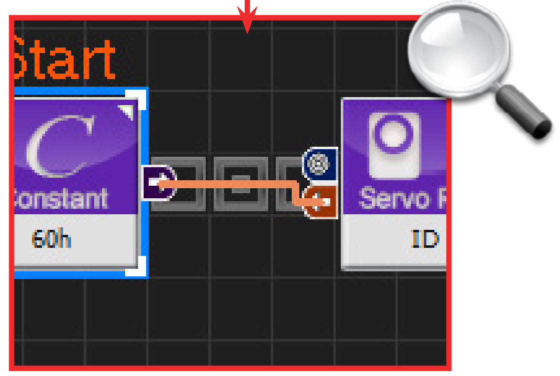
```
void main()
{
        SERVO_ID[254]=0x60
        jog( 512, 0, 254, 100 )
        delay( 1000 )
        jog( 235, 0, 0, 100 )
        delay( 1000 )
        jog( 235, 0, 1, 100 )
}
```

08 Viewing C-Like

From the Graphic tab on the top right, click on the "C-Like" tab to open the program source window as shown in the diagram to the left.
This window shows the source codes of the entire program. Since codes are very similar to the C language structure, studying the codes will help the user become familiar with the C language structure. When module is clicked, cursor in the C-Like window jumps to the text line showing the source codes of the module.

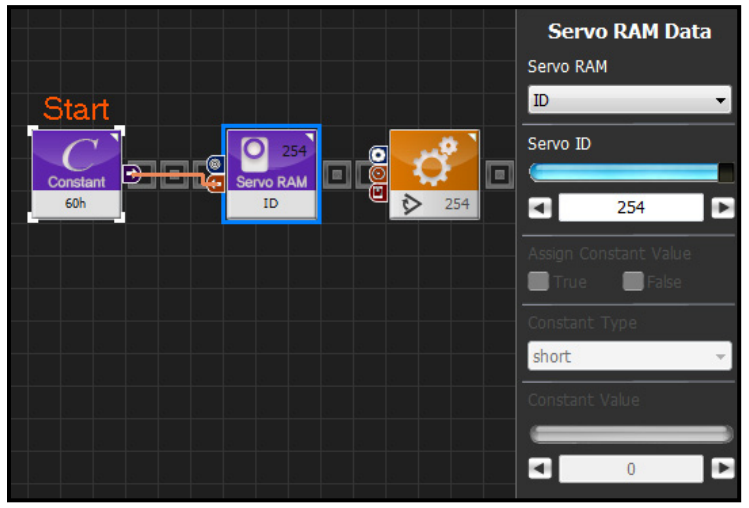


Constant
Constant Type: int
Constant Value: 0x60



09 Set Constant

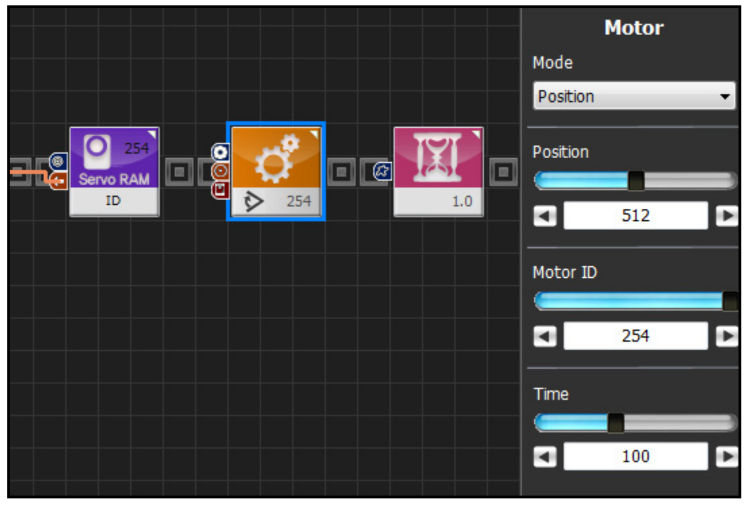
This section allows the servo motor to operate on its own. From the Constant module properties, click on the Constant Value and change the value to 0x60. Constant with Hex number of 0x60 shows the torque enabled state of the servo motor. This value is sent to the input pin of the next module through the output pin.



Servo RAM Data
Servo RAM: ID
Servo ID: 254
Assign Constant Value: True False
Constant Type: short
Constant Value: 0

10 Apply to All Servos.

This section applies constant value 0x60 received from the previous section to all servo motors.
Select Data > Servo RAM and place it after the Constant module.
Servo RAM : Select TorqCtrl
Servo ID : Set to 254, ID 254 means setup will be applied to all connected servos.
Use the Connector to connect the output pin of the Constant module to the second input pin of the Servo RAM module.

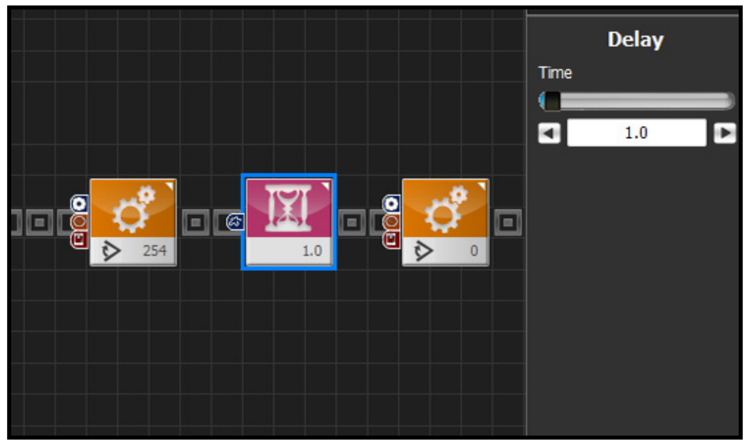


Motor
Mode: Position
Position: 512
Motor ID: 254
Time: 100

11 Set Position to All Servos

This section sets all servo motor positions to the center. Centering the servo motors result in the robot stretching out both arms to each side.

Select Motion > Motor and place it after the previous module.
Set Mode : Position, to control the angle.
Set Position : 512, Position value of 512 will send the motor to the 0 point (center).
Set Motor ID : 254, ID 254 will apply the setup to all connected servo motors.
Set Time : 100, 1 unit = 11.2ms, value of 100 is approximately 1.12s. Motors will be sent to the desired position in 1.12s.

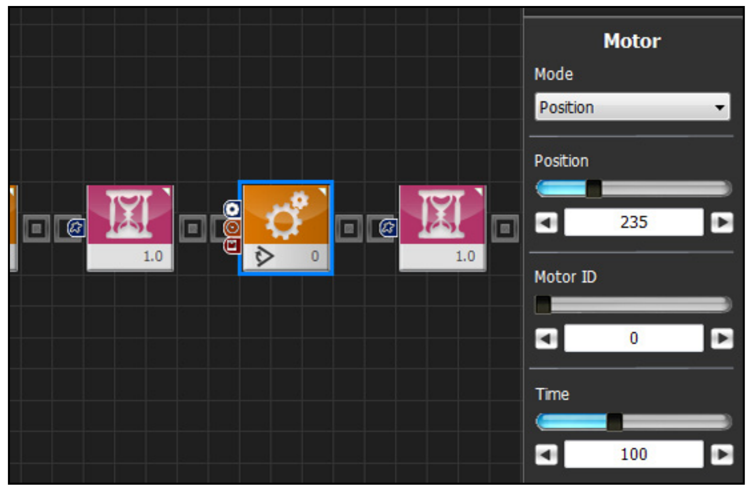


Delay
Time: 1.0

12 Delay

This section adds 1s of delay before starting the next motor operation. Delay allows the motor to move to the position requested by the previous module without any interruption.

Select Flow > Delay module and place it after the previous module.
Set Time : 1.0, delay 1s.

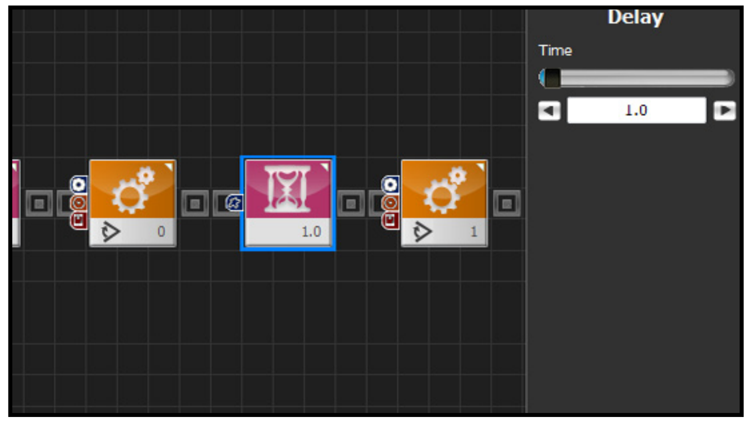


Motor
Mode: Position
Position: 235
Motor ID: 0
Time: 100

13 Set Motor ID 0 (Right Shoulder)

This section turns the right shoulder in order to lower the arm to the side of the body.

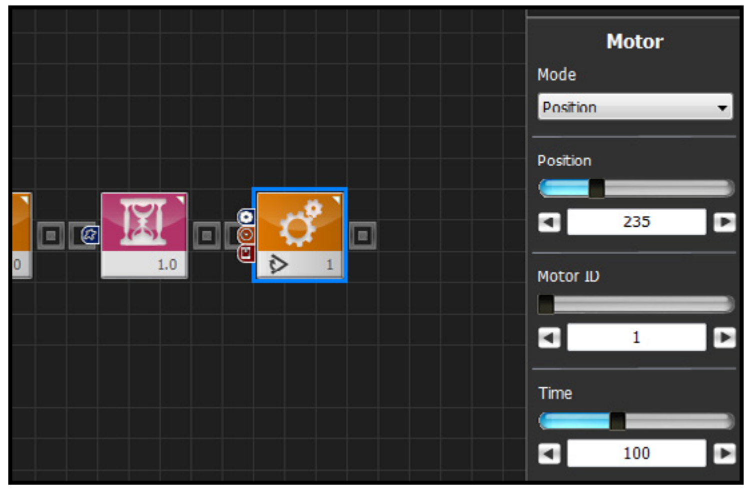
Select Motion > Motor module and place it after the previous module
Set Mode : Position to control the motor position.
Set Position : 253. Changing the motor position to 253 will allow the horizontally stretched arm to be lowered to vertical position.
Set Motor ID:0, right shoulder motor has ID 0.
Set Time:100, motor will move to the required position in approximately 1.12s.



Delay
Time: 1.0

14 Delay

This section adds 1s of delay before starting the next motor operation. Delay allows the motor to move to the position requested by the previous module without any interruption.
Select Flow > Delay module and place it after the previous module.
Set Time : 1.0, delay 1s.

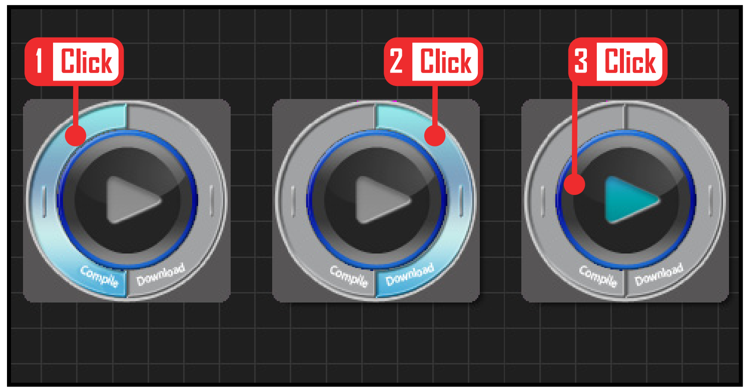


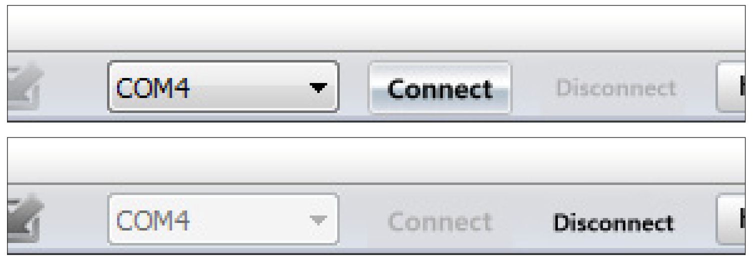
Motor
Mode: Position
Position: 235
Motor ID: 1
Time: 100

15 Set Motor ID 1(Upper Right Arm)

Select Motion > Motor module and place it after the previous module.

Set Mode : Position to control the motor position.
Set Position : 235. Changing the motor position to 235 will lower the horizontally stretched arm to the new vertical position.
Set Motor ID : 1, upper right arm motor has ID 1.
Set Time: 100, motor will move to the required position in approximately 1.12s.

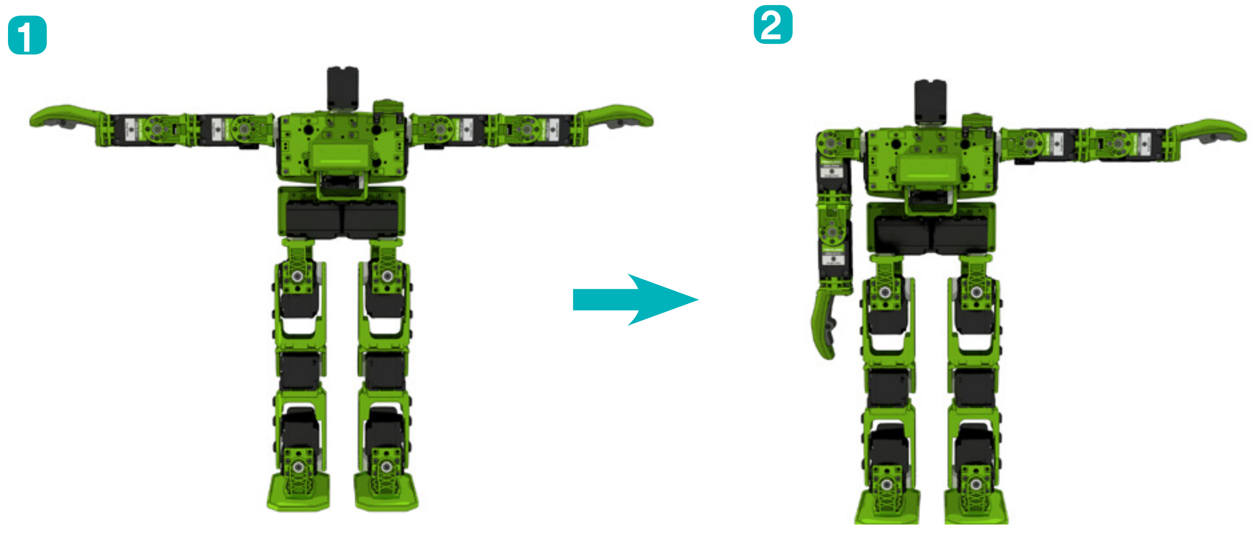


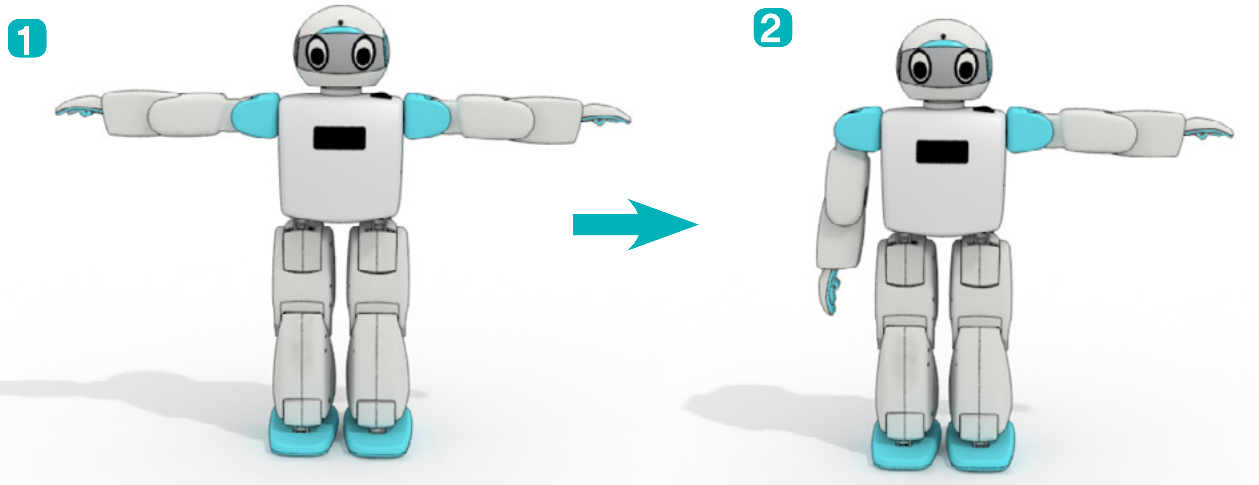


16 Download

After programming is completed.
Compile -> Download -> Run.

Select the appropriate COMPORT or USB-to-Serial number located on the PC and press Connect to open the serial port.
Click 'Compile' and then click 'Download' on the right to download the program to the robot if no compile error is found.
Click "Run" button (arrow in the middle) after completing the download.

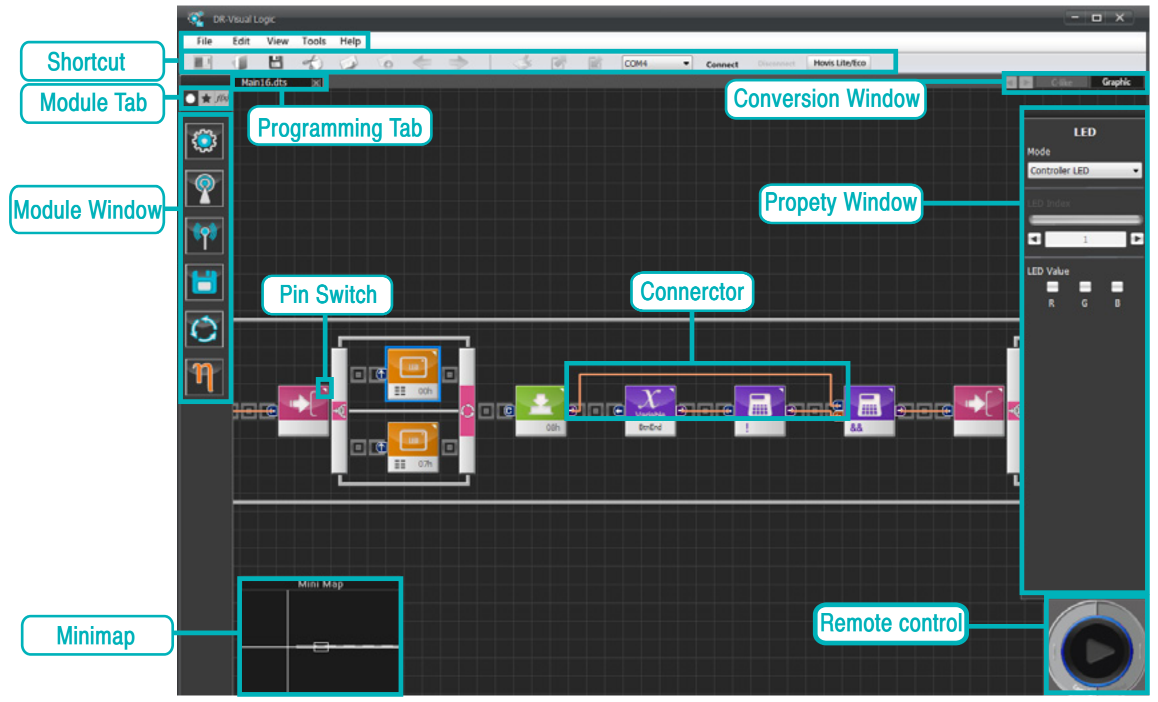




17 Robot Motion

Robot will lower the right arm to side of the body.

User Interface 03



1 **Menu :** Composed of File/Edit/View/Tools/Help. Allows user to use DR-Visual Logic functions.

2 **Tools :** Collection of frequently used function icons.

3 **Module Tab :** Tab for selecting Module Bar. Module Bar is composed of View/Favorite/My Module.

4 **Module bar :** Bar located on the left side of the window is for selecting and adding modules. View Bar is composed of 5 types of module packs Motion / Sensor / Communication / Data / Flow. Hovis Lite/Eco has an additional third party module pack (ETA).

5 **Programming Tab :** Enables the user to see the name of the file currently being edited and also enables user to switch between the windows when more than one file is being edited at the same time.

6 **Switch Button :** Composed of buttons for moving L/R to view the desired programming tab when number of open programming tabs go beyond the screen width. Also contains button for switching between the graphic programming window and the "C-Like" window containing text source codes.

7 **Property Window :** Window for entering module properties. Each module has variety of properties which can be changed and set from the property window. Some of the property values can be entered using the input pin as well.

8 **Remote control :** Simple remote control like buttons for giving Compile/Download/Run commands.

9 **Mini Map :** Shows the shape of the entire program, current location, and enables the user to jump to any spot on the program by simple click.

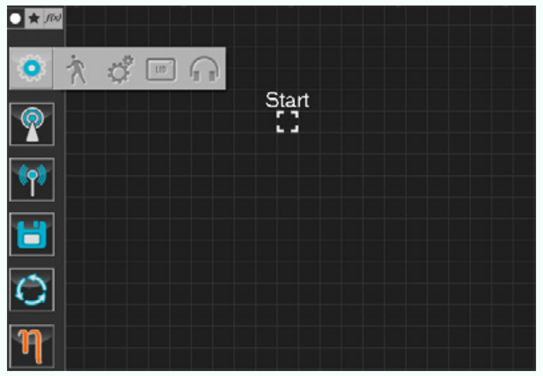
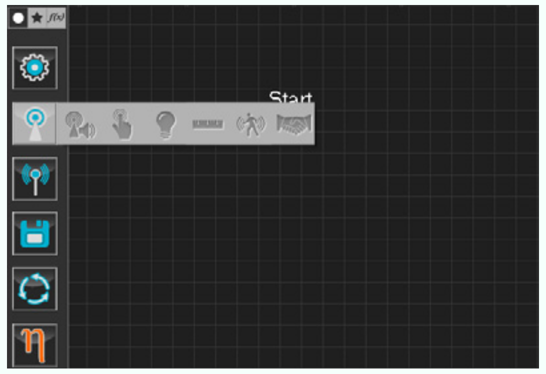
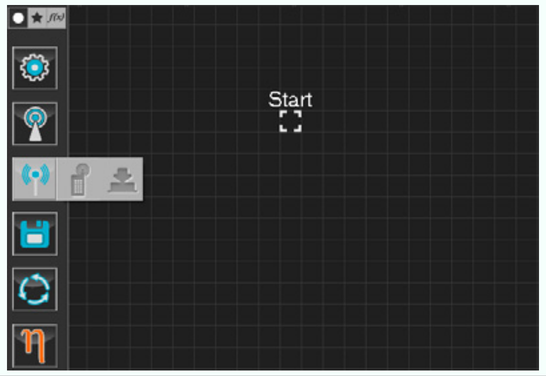
10 **Pin :** Composed of Input and Output pins. Input/Output pins are used when output value of one module will be used as an input value of another module.

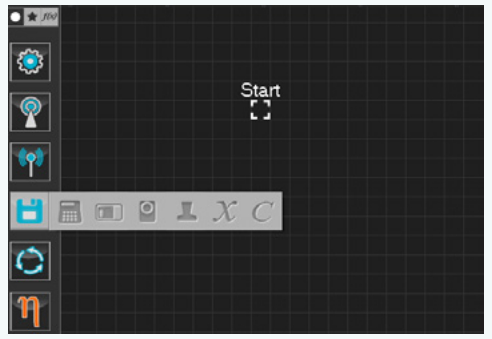
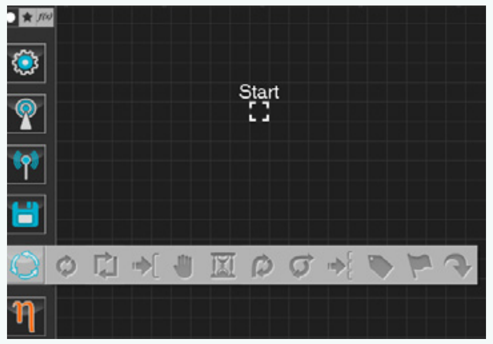
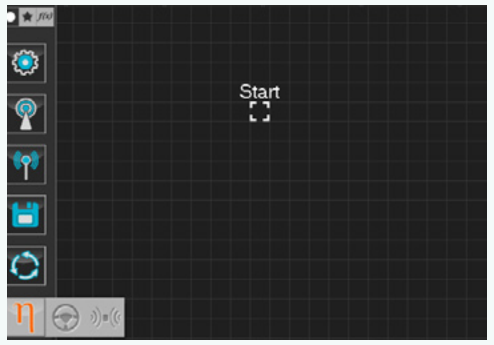
11 **Pin Switch :** Used to show the name of the pin. Click once to make the pin name appear and click once more to make the name disappear.

12 **Connector :** Connecting line for connecting pin to pin.

04 Programming Module

DR-Visual Logic is composed of following modules.
Module packs contain all the programming modules required to create a program. Each module is composed of a function supported by either the controller DRC (HOVIS Lite/Eco) or by the MID (HOVIS Genie). This manual covers only the modules applicable to HOVIS Lite/Eco. Please refer to the HOVIS Genie manual if HOVIS Genie was selected when new file was being created.

Module Pack	Picture	Module	Description
Motion		Move	Run saved robot motion
		Motor	Position/speed control by each motor
		LED	Head LED - Run saved head LEDController LED - Control LED on the controller
		Sound	Melody - Run saved Buzzer melody Note - Run single buzzer note
Sensor		Sound Sensor	Internal, distinquishes Left & Right
		Touch	Recognize touch to head module (Used when head module is installed)
		Light	Measures light (Internal)
		Distance	Measures distance (Used when distance sensor is installed in the controller)
		Dynamics	Measures acceleration and angular velocity (Used when accelerometer/ Gyro is installed in the controller).
		Hand Touch	Recognizes press to the palm tact switch (Used when palm switch is installed at the end of the arm)
Com-munication		IRRceiver	Recognize remote control data
		Button	Reconginze rear controller button.

Module Pack	Picture	Module	Description
Data		Operator	Operator(Arithmetic/Logical/Relational/Bit-wise/Increment/Decrement)
		MPSU RAM	Used to R/W DRC-005T RAM register value
		Servo RAM	Used to R/W DRS-0101/0201 servo motor RAM register value.
		OPSU RAM	Use to R/W RAM register value when DRC-004TO (Omniwheel drivetrain sensor module) is installed.
		Variable	Use when user variable is being used.
		Constant	Use when entering constant value
Flow		Loop	Infinite loop / for statement
		While	While statement (repeat when specific condition is True)
		If-else	if-else statement (Branch control)
		Wait	Wait during specific condition
		Delay	Wait for specified time
		Continue	Go back to the beginning of the loop statement
		Break	Exit from the loop or switch-case statement
		Switch	Switch in the switch-case statement
		Case	Case in the switch-case statement
		Label	Assign specific location of the program as label
		Goto	Move to assigned label.
ETA		ETA Actuator	Use when Actuator from the third party module series ETA is connected
		ETA Sensor	Use when Sensor from the third party module series ETA is connected.

Variable Type of Variable Modules

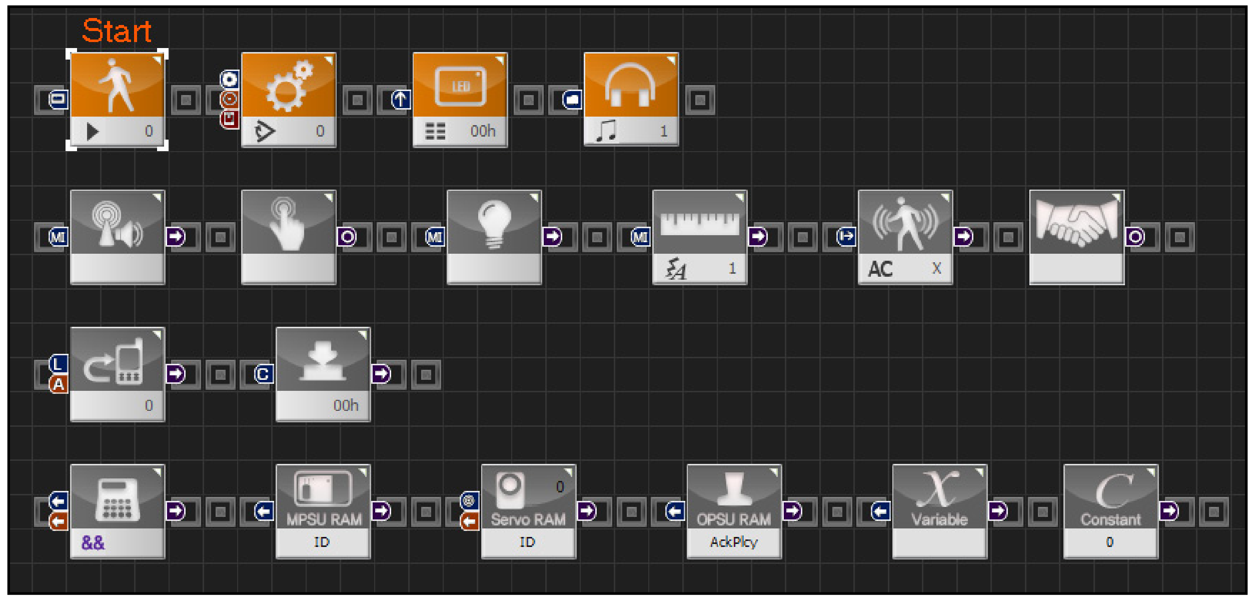
Variables are used to declare user variables. There are different types of variables with different names according to the type and range of the data being saved. DR-Visual Logic variables are same as the variables in C/C++ with identical names. User variable is declared automatically when it is first used and has initialization value of 0 unless another initialization value was specifically entered. Limited memory space in DRC-005T can be utilized optimally by declaring the variable type with appropriate data range according to the estimated variable value.

Type	Range	Description
bool	True(1)/False(0)	True/False
char	-128~127	1 Byte signed integer
unsigned char	0~255	1 Byte unsigned integer
short	-32,768~32,767	2 Byte signed integer
unsigned short	0~65,536	2 Byte unsigned integer
int	-2,147,483,648~2,147,483,647	4 byte signed integer

04-1 Programming Module Regular Module

Regular modules are connected together and used sequentially.

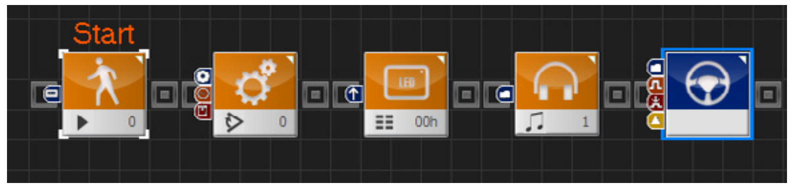
All modules except for the flow modules are regular modules.



From the top. module icons represent Motion, Sensor, Communicaiton, and Data.

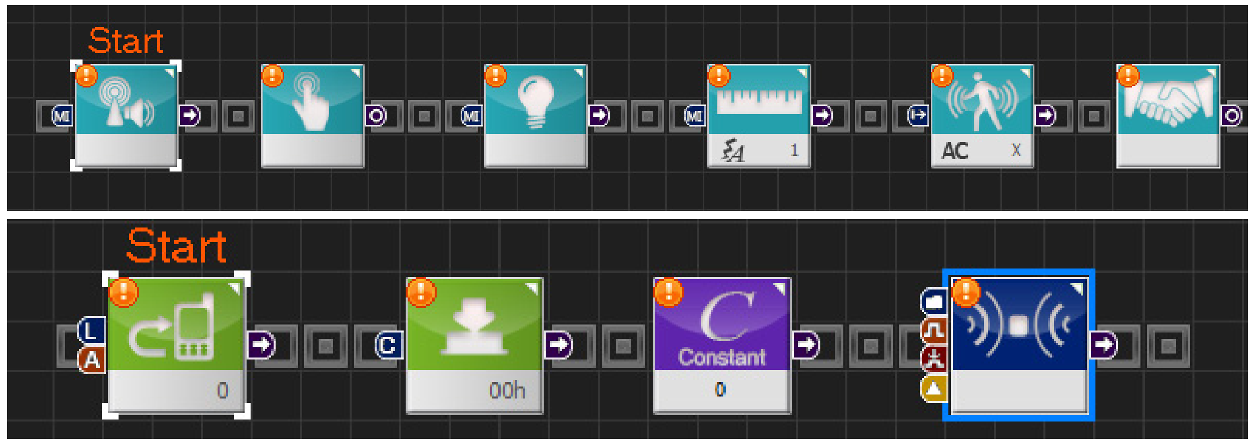
Using Regular Modules

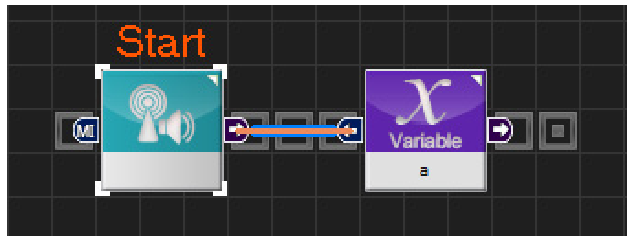
Some modules can be used independently while other modules require connection to other modules through the out-put pin to be activated.



```
void main()
{
          motion( 0 )
          jog( 512, 0, 0, 60 )
          led( 0x00 )
          melody( 1 )
          ETAAct( 253, 0, 0, 0 )
}
```

Modules in the Motion module pack and ETA Actuator in ETA does not have an output and generate source codes even when used independently.

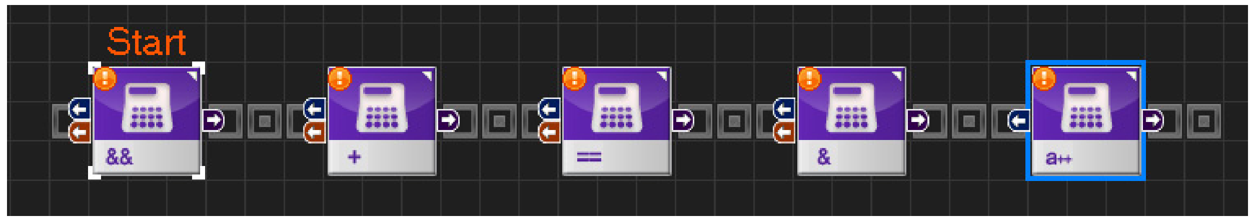




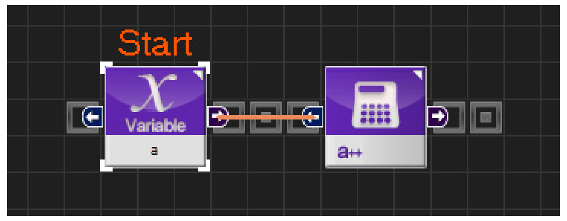
```
bool a
void main()
{
          a=( MPSU_SoundDetectFlag && MPSU_SoundDir
}
```

Modules in the Sensor module pack and Communication module pack as well as Constant in Data and ETA Sensor in ETA are not able to generate source codes independently. Fore mentioned modules require output pin of the module to be connected to the input pin of other module in order to generate the source codes. Error is shown on the top left side of the module when module is not able to generate source codes independently.

Depending on the circumstances, modules in the Data module pack may or may not be able to generate source codes independently.

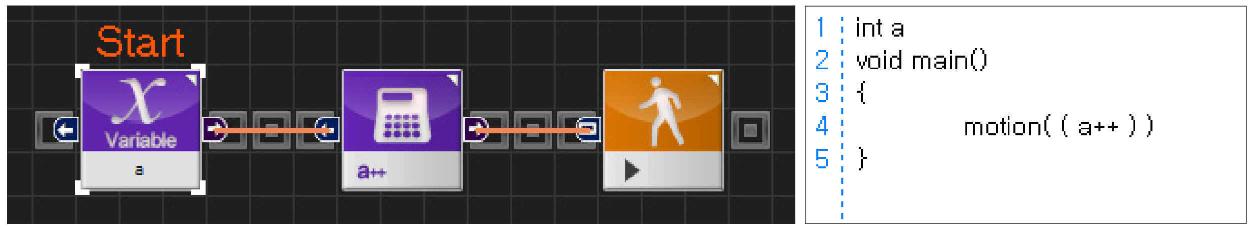


Most of the Operator modules require output pin of the module to be connected to an input pin of another module in order to generate source codes.



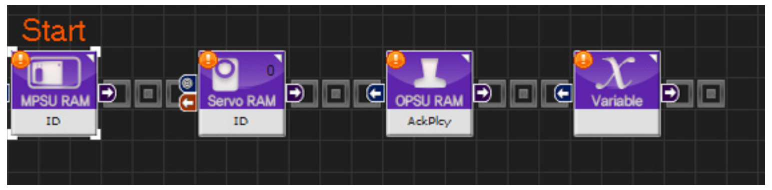
```
int a
void main()
{
          a++
}
```

One exception is when variable capable of changing the value is connected to the Incremental, in which case source codes increasing the value of variable are generated even without connection to the output pin.

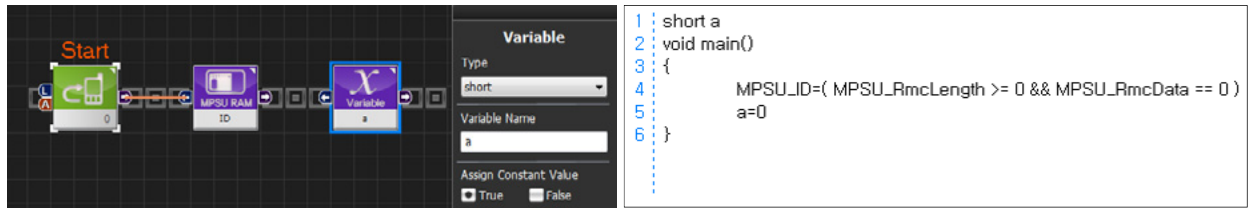


```
int a
void main()
{
            motion( ( a++ ) )
}
```

When output pin of the module is connected to another module under the same circumstances, source increasing the value of the variable becomes incorporated into the source of the connected module.



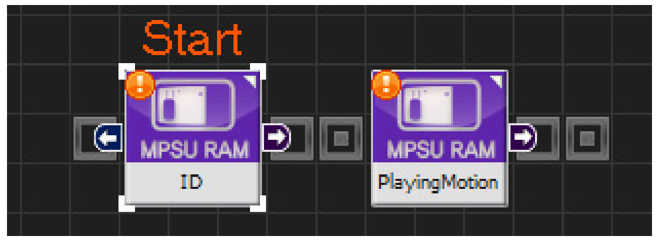
MPSU Ram, Servo RAM, OPSU Ram, Variables do not generate source codes independently when first created.



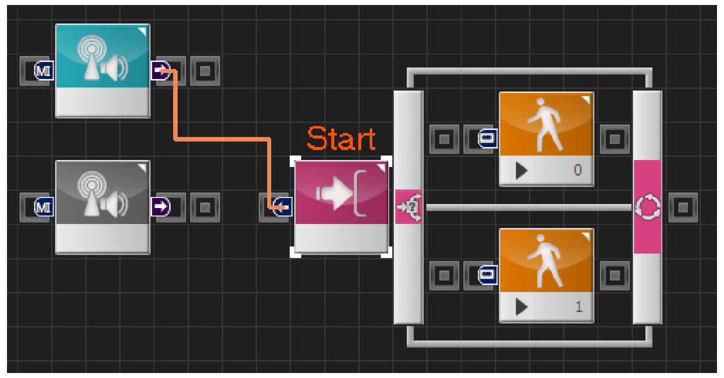
Variable
Type: short
Variable Name: a
Assign Constant Value: [✓ True] False

```
short a
void main()
{
            MPSU_ID=( MPSU_RmcLength >= 0 && MPSU_RmcData == 0 )
            a=0
}
```

However, when another module is connected to the input pin, source codes are generated to substitute the value of the variable even without connection to the output pin. Also, codes are generated to substitute the value of the constant below when Assign Constant Value in the property window is checked True.



Also, among the MPSU RAM, Servo RAM, and OPSU RAM, there are items such as the sensor value and location data in which value can only be read from and not written to. In such a circumstances, input pin disappears and substitution is not possible.

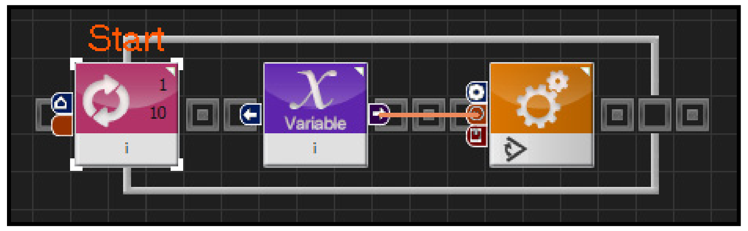


Modules that are able to generate source codes even without connection to the output pin must be placed on the program line that starts from the Start in order to become active and generate source codes. However, those modules that require connection to the output pin in order to generate source codes are not required to be placed on the program line as long as the connector connection exists.

Programming Module

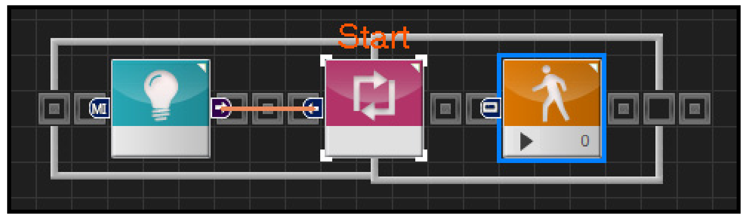
Flow Module 04-2

Flow type modules connect to the regular modules and control the flow of the program using loop, switch, and etc. Unlike regular modules, flow type modules have graphic structure with outline appearing around the module.



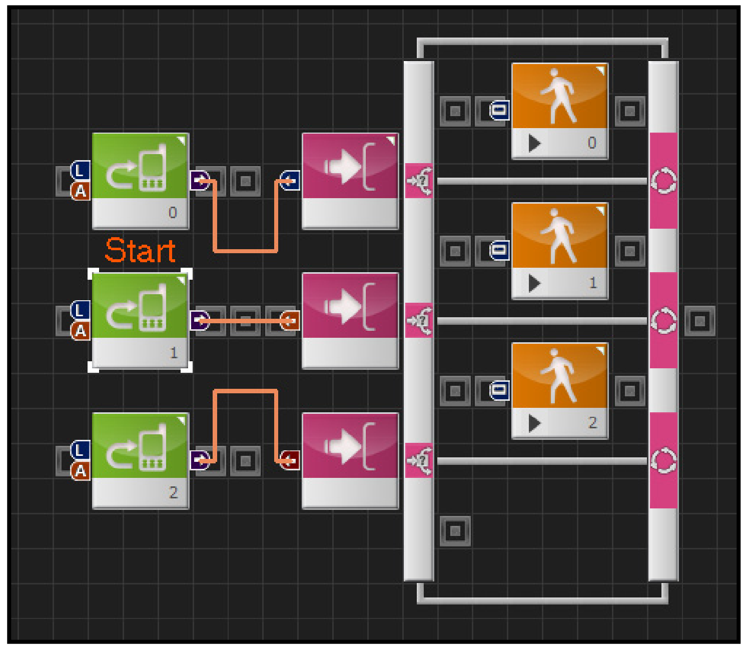
Loop

Loop module commands repeat of certain section of the program. Module contains For statement which repeats the section specified number or times or Forever statement which repeats the section infinitely.



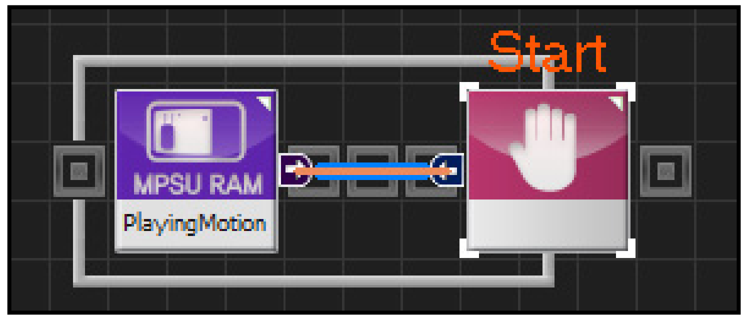
While

While module processes the following section as long as the condition is True. It is a loop statement with attached condition.



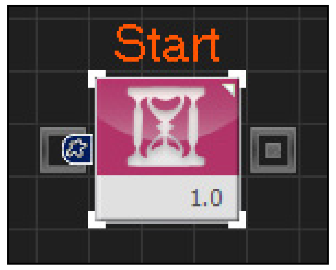
If-else

Depending on the condition, If-else statement branches the program. Number or branch statements can be increased or decreased by clicking on the button in the property window.



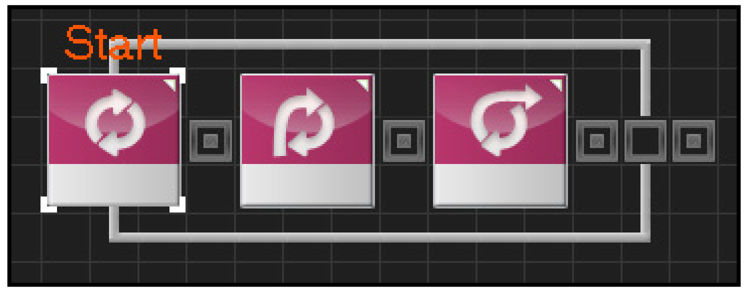
Wait

Pause program execution while input condition is true and continue again when condition becomes false.



Delay

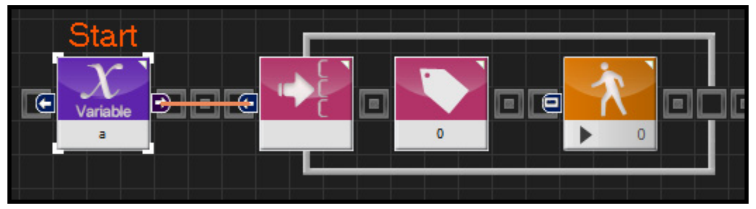
Delay program execution for specified period of time.



Continue, Break

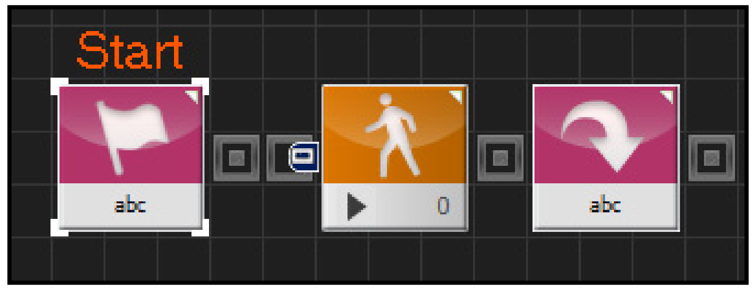
Continue sends the program back to the beginning of the loop and can only be used within the loop statement.

Break is used to exit from the loop and can only be used within the loop and switch-case statement.



Switch, Case

These 2 modules are used together and form switch-case statement. Depending on the switch module input, program after the corresponding case statement will be executed.



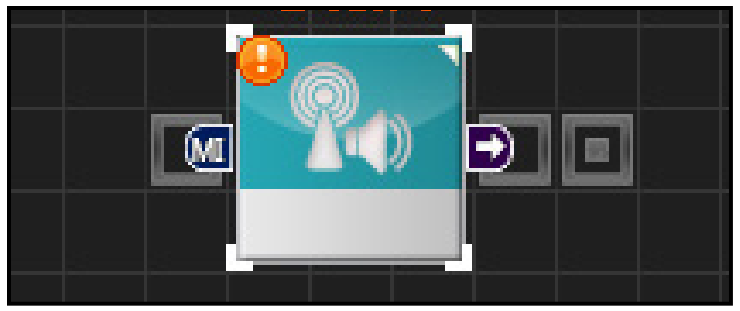
Label, Goto

These 2 modules are used together to create Jump function. When goto statement with the same name as the label module is created, program will jump to the label module when goto stamen is met.

Programming Module

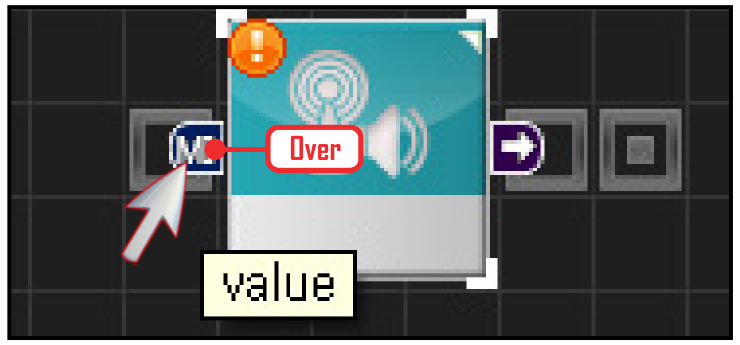
Pin 04-3

Some modules have input and output values. Resulting value of the output pin connected to an input pin of another module becomes an input value of that module.



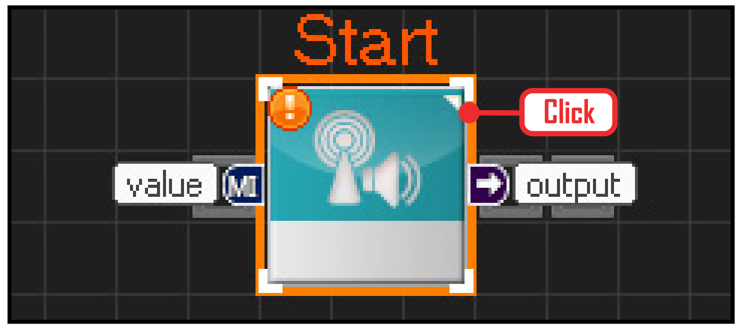
Pin

Modules with input/output values have pins located on the left and right side of the module. Pin on the left side is the input pin and the pin on the right side is the output pin.



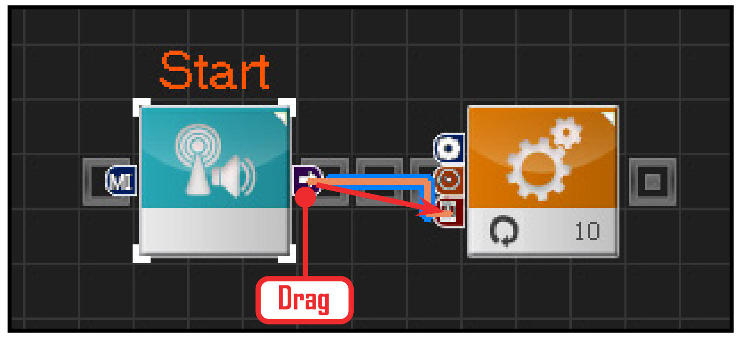
Help Balloon

It is difficult to distinguish the connector just by looking at the connector icon. To find out the function of the connector, place the mouse cursor on top of the connector and balloon will appear with the name of the connector.



Opening Help Ballons

To view the name of several pin names at once, click on the triangle shaped pin switch at top right corner of the module. Pin names will appear beside each pin and disappear when switch is clicked once more.

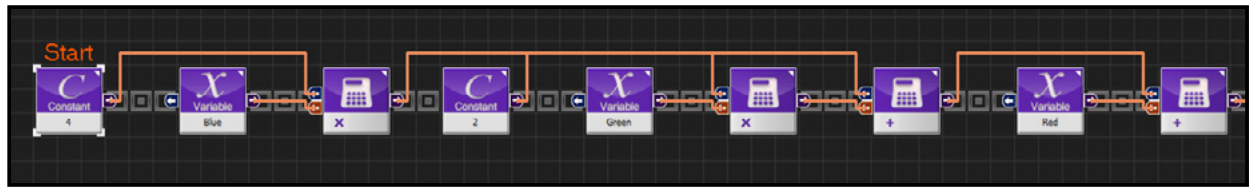


Pin Connection

In order to enter the output value of the first module as the input value of the following module, use the mouse to connect two pins as shown in the diagram to the left.

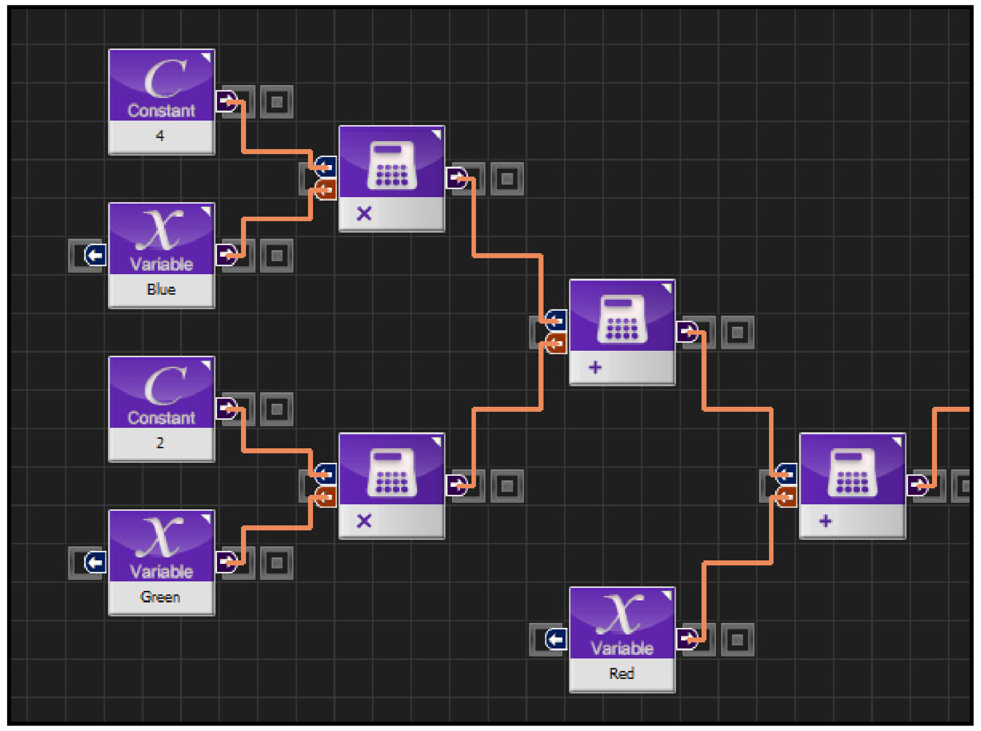
04-4 Programming Module Connection Type

Module connections can be either serial type connection or row type connection.



Serial Type Connection

In serial type connection, modules are connected sequentially from left to right. The photo above shows arithmetic calculation program. ((4xBlue)+(2xGreen))+(1xRed) calcuation shown as serial type connection.

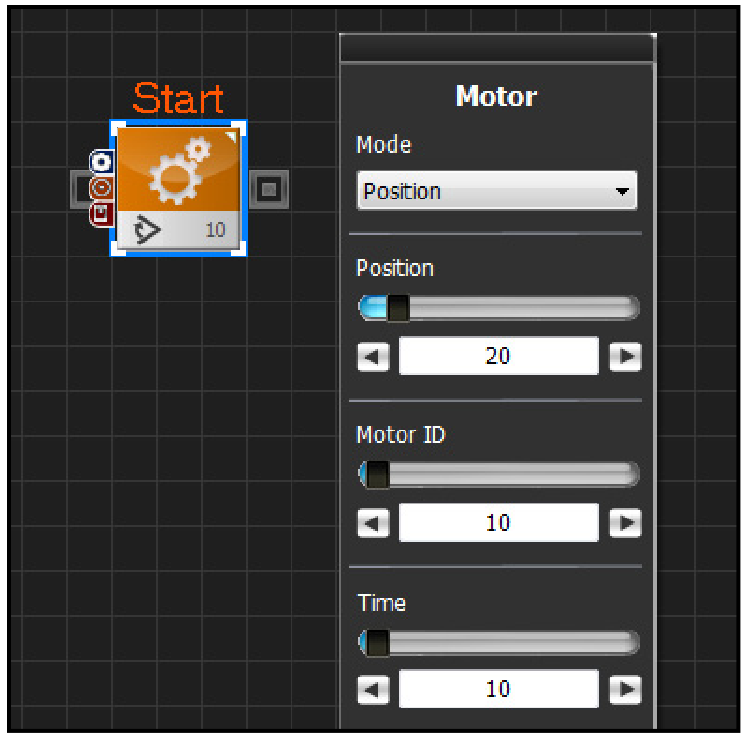


Row Type Connection

Row type connection, modules are connected in rows using vertical spacing. The 2nd photo with row type connection is same program as the 1st photo with serial type connection.

05 Property Window

Modules have their own properties and thses propeties must be given a value for program to work. UI in property window includes list popup, radio button, number setting, and etc. Refer to the Help file for details on properties for each module, property values, and limits.



Motor

Mode: Position

Position: 20

Motor ID: 10

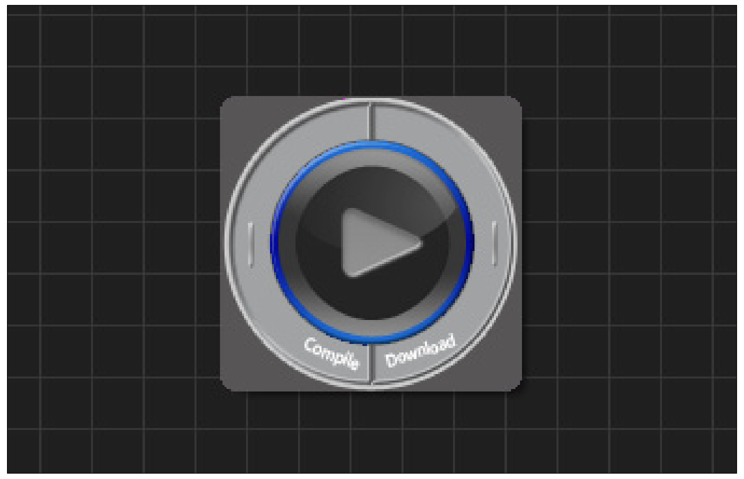
Time: 10

Property Window

Modules have their own properties and thses propeties must be given a value for program to work. UI in property window includes list popup, radio button, number setting, and etc. Refer to the Help file for details on properties for each module,
property values, and limits.

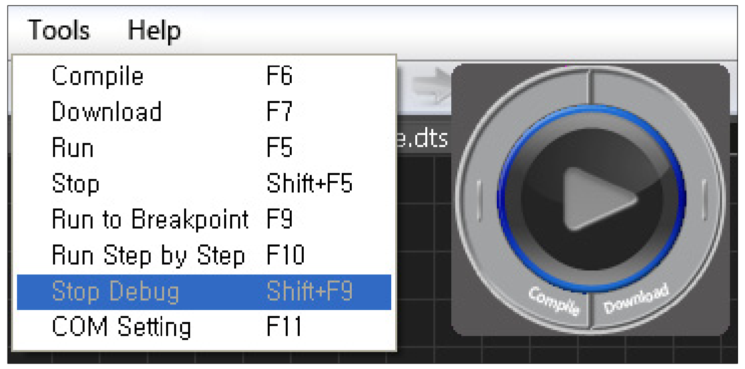
06 Compile/Download

Once the programming is complete, it is compiled, downloaded to the robot and run. Downloader is a large icon located at bottom left side of the programming window. More specific commands are found in the tools menu.



Downloader Icon

Downloader icon has three commands.
Compile command on the left, download command on the right, and play command in the middle shown by arrow like icon.



Tools Menu

Commands in the tools menu allow the user to give more detailed and specific commands.
Compile : Compile completed program.
Download : Download compiled program
Run : Run downloaded program
Stop : Stop the program execution
Run to breakpoint : Assign breakpoint in the source codes from the C-Like window to pause the program at assigned location.
Run in single steps : Runs the source codes in the C-Like window one line at a time.

Various Functions 07

The following table idetifies menu functions provided by DR-Visual Logic.

Type	Name	Description
File	New Window	Create new dts file.
	Load	Load saved dts file.
	Close	Close open dts file.
	Save	Save current dts file.
	Save As	Save current dts file under another name.
	Print Preview	Preview on screen prior to printing.
	Print	Print.
	Quit	End program.
Edit	Undo	Undo previous command.
	Redo	Redo previous command.
	Cut	Cut selected section to the clipboard.
	Copy	Copy selected section to the clipboard.
	Paste	Paste content of the clipboard.
	Delete	Delete selected section.
View	Default View	Change to default view.
	Zoom In	Zoom In
	Zoom Out	Zoom Out
	Center This Module	Move the screen with selected module as the center. Move to Start point if module has not been selected. Click on the source code line in the C-like window and use this function to find the matching module.

Type	Name	Description
Tools	Compile	Compile completed program.
	Download	Download compiled program.
	Run	Run downloaded program.
	Stop	Stop program execution.
	Run to Breakpoint	Pause program at the break point designated in the C-Like window.
	Run Step by Step	Run the source codes in the C-Like window one line at a time
	Stop Debugging	Stop debugging.
	Comm Setting	Set communication setting. Select between Serial and Wifi. Select Serial for Lite/Eco.
	Controller Error Check	Check current DRC-005T for error.
Help	Index	Open help file
	Online Help	Open Dongbu Robot website.
	S/W Update	Check software for latest version
	F/W Update	Update controller firmware.
	F/W Restore	Restore controller firmware if controller does not function or wrong firmware was installed.
	About	Open about program window

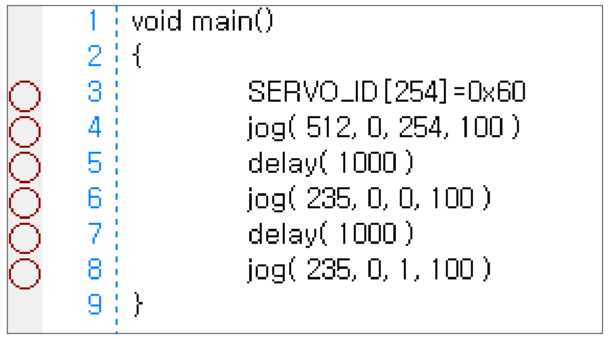
Miscellaneous Functions

Click when button and drag: moves the screen

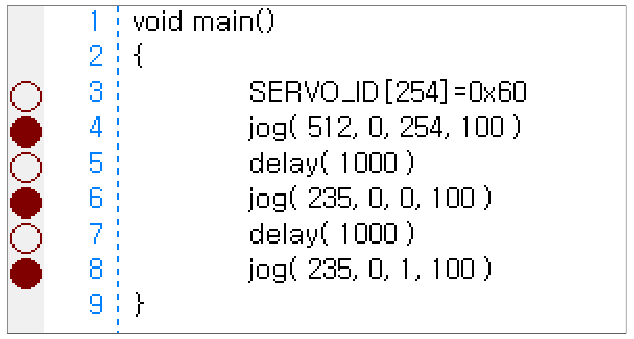
Shift + left click and drag: moves the screen

Ctrl + left click and drag selected module : copies selected module

Debugging

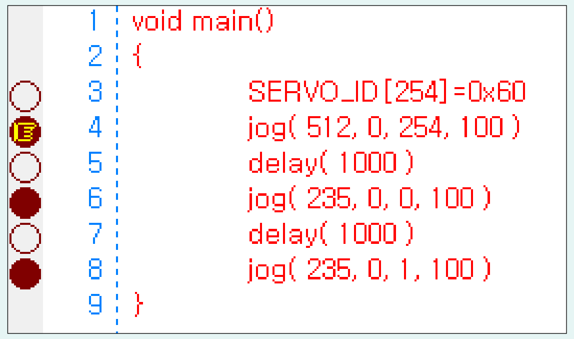


```
void main()
{
        SERVO_ID[254]=0x60
        jog( 512, 0, 254, 100 )
        delay( 1000 )
        jog( 235, 0, 0, 100 )
        delay( 1000 )
        jog( 235, 0, 1, 100 )
}
```



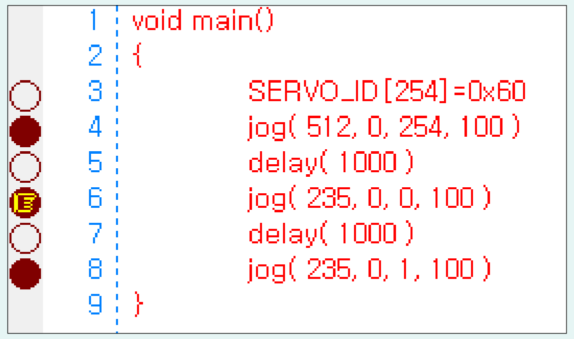
```
void main()
{
        SERVO_ID[254]=0x60
        jog( 512, 0, 254, 100 )
        delay( 1000 )
        jog( 235, 0, 0, 100 )
        delay( 1000 )
        jog( 235, 0, 1, 100 )
}
```

After programming is complete, press C-like button to view the generated source codes.
Select a location in the C-like window and press the round button on the left side to assign breakpoint. When "Run to Breakpoint" or "Run Step by Step" command is issued from the Tools, current program will be compiled, downloaded, and then executed in debugging mode.



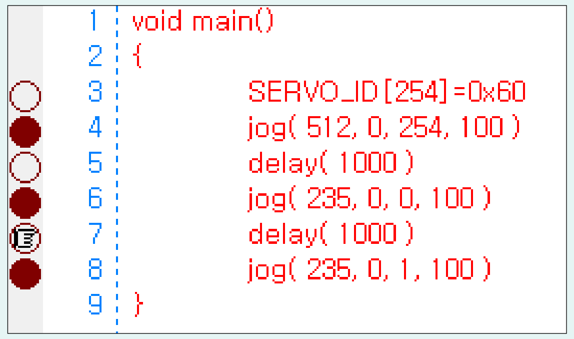
```
void main()
{
        SERVO_ID[254]=0x60
        jog( 512, 0, 254, 100 )
        delay( 1000 )
        jog( 235, 0, 0, 100 )
        delay( 1000 )
        jog( 235, 0, 1, 100 )
}
```

Executing "Run to Breakpoint" command. Debugging started, program stopped at the first breakpoint.



```
void main()
{
        SERVO_ID[254]=0x60
        jog( 512, 0, 254, 100 )
        delay( 1000 )
        jog( 235, 0, 0, 100 )
        delay( 1000 )
        jog( 235, 0, 1, 100 )
}
```

Program will stop at the next break point if "Run to Breakpoint" command is issued again.



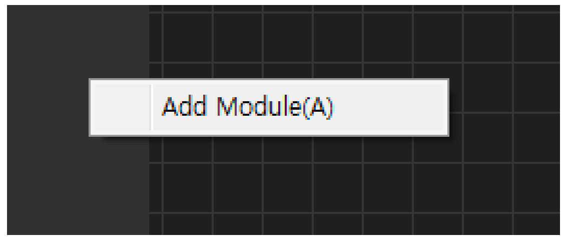
```
void main()
{
        SERVO_ID[254]=0x60
        jog( 512, 0, 254, 100 )
        delay( 1000 )
        jog( 235, 0, 0, 100 )
        delay( 1000 )
        jog( 235, 0, 1, 100 )
}
```

If "Run Step by Step" command is issued, program will stop at the next line regardless of the breakpoint.

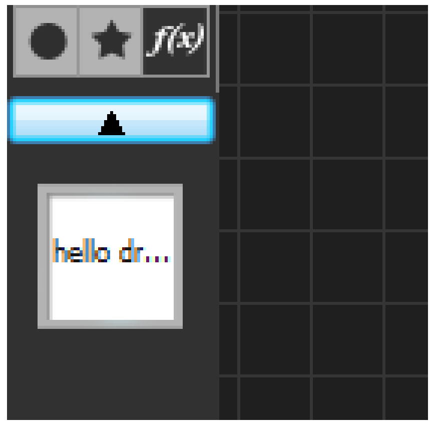
Debugging function allows the user the check the progress of the program when program does not function as expected.

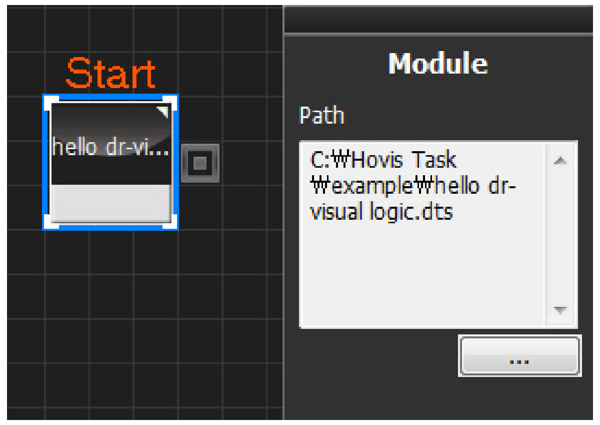
Using My Module

dts file created by DR-Visual Logic can be called from another dts file and used like a function.



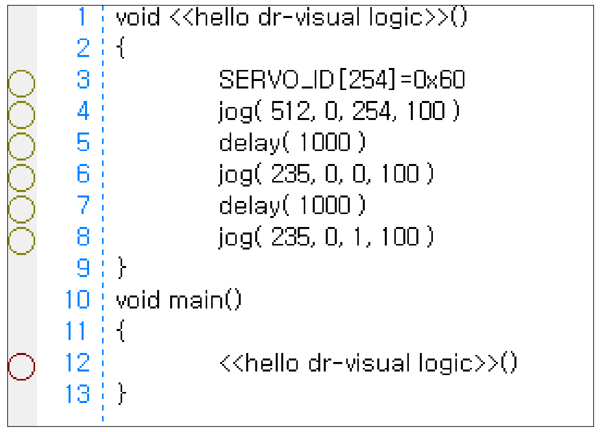
Select My Module from the module tab and then right click to issue add module command. Select and add dts file from the selection window.





C:₩Hovis Task ₩example₩hello dr-visual logic.dts

My Module added. Click and place the dts file to use it as modularized function.



```
void <<hello dr-visual logic>>()
{
        SERVO_ID[254]=0x60
        jog( 512, 0, 254, 100 )
        delay( 1000 )
        jog( 235, 0, 0, 100 )
        delay( 1000 )
        jog( 235, 0, 1, 100 )
}
void main()
{
        <<hello dr-visual logic>>()
}
```

Switch to C-Like window to view the source codes and notice how the added module is being used like a function.

Programming Individual Modules 08

Provided sample program is based on 16 axis humanoid robot with DRC controller platform. Sample program will require reprogramming if it is to be used for 18,20 axis humanoid robot or other variations with change in modules or motions.
Before running the program, check the motor ID and robot sensor locations. Also, use the DR-SIM to check the saved motion list and apply correct index values. Provided sample program is as follows.

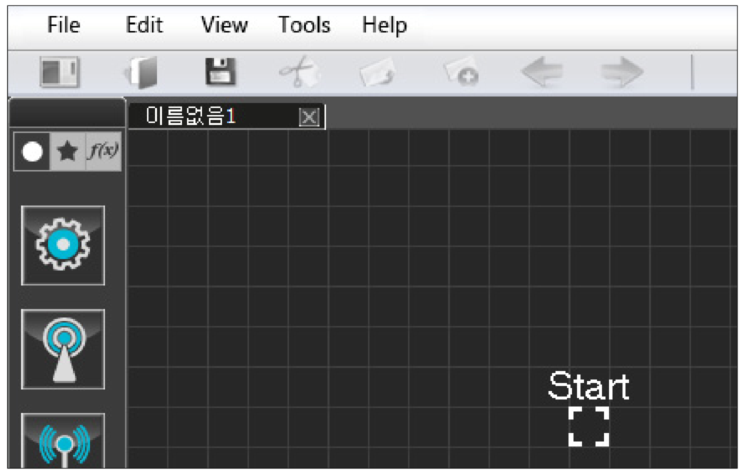
Module Pack	Module	Example
Motion	Move	Move module loads the robot motion saved in the DRC controller and applies it to the program. Robot motion can be loaded by the number, and if required, names can be checked from DR-SIM. This program will repeat running the motion creatd by DR-SIM on the robot indefinitely. This is a relatively complicated program useful for reliability test or for demostration purposes.
	Motor	This program creates dancing motion by controlling individual motors.
	LED	This program will turn on/off the LED by pressing the button on DRC Controller. (With Button)
	Sound	This progam will output sound when input from remote control buttons(#1~8) is received. (With IRRecieve)
Sensor	Sound Sensor	Sound sensors are located inside the DRC on both sides. This program will make the robot respond to the left clap by lifting the left arm and to the right clap by lifting the right arm (Sample # 2). Robot may have difficulty distinquishing the direction of the clap when there is lots of background noise. It may respond by lifting both arms to a single clap from one direction or respond erratically. More refined programming is required to make the robot to respond more reliably regardless of the background noise. Refining the program by forcing a DELAY after registering the first sound so that it will not receive anymore sound input will increase the reliability.
	Light	This program makes the motor respond to the external luminosity. When luminosity decreases, robot will lift up the left arm (Covering the CDS sensor at back of the controller will decrease the luminosity and robot will respond by lifting the left arm).
	Distance	PSD Digital(Distance Sensor) : This program makes the robot walk backwards, turn right, and then walk straight if it detects a wall within certain distance. PSD Analog(Distance Sensor) : This progam makes the robot turn left to avoid the wall.
	Dynamics	Accerlaration : This program makes the robot stand if it falls forward, makes the robot stand if it falls backward.
Communication	IRRciever	This program assigns different sound notes to the remote control buttons 1~8 and makes the DRC controller play the sound.Buttons1~8 matches Do~Si. (With Sound)
	Button	LED at back of the DRC respond to the press of a button on DRC (with LED)

Programming Individual Module

Move 08-1

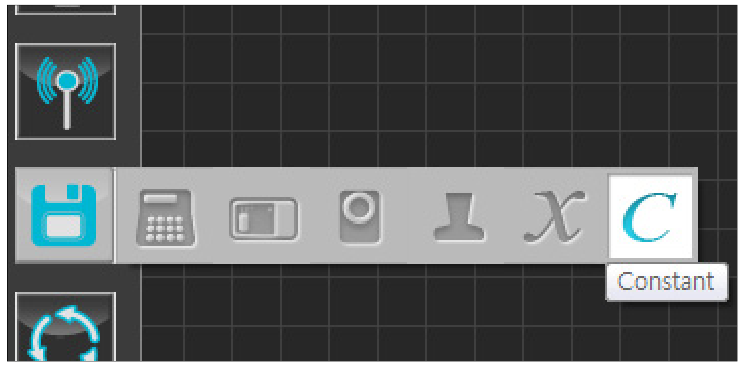
Example Description

Move module loads and runs the robot motion saved in the DRC controller. Robot motion can be loaded by the number, and if required, name of the motion corresponding to motion number can be found in DR-SIM. This program will continuously repeat the robot motion created and downloaded from DR-SIM. Completed program is useful in testing the reliability of the robot or as demonstration program.



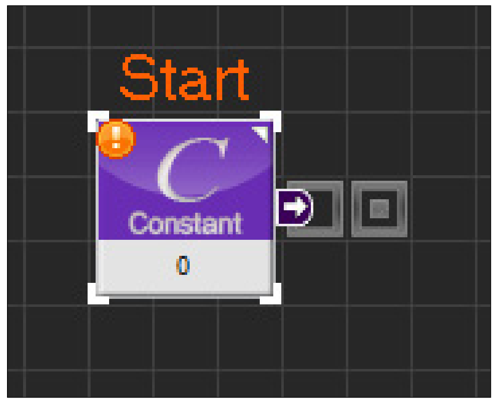
01 Assign Variable

Click on the left most icon in the tools to create new dts file. Select HOVIS Lite/Eco as the Robot.



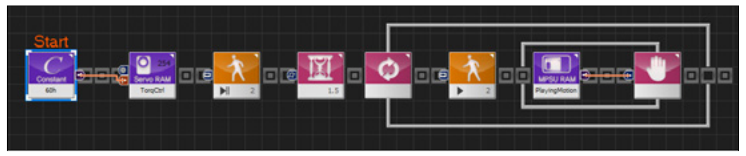
02 Select Module

Prior to placing the module, click to select the module from the module bar located on the left side.



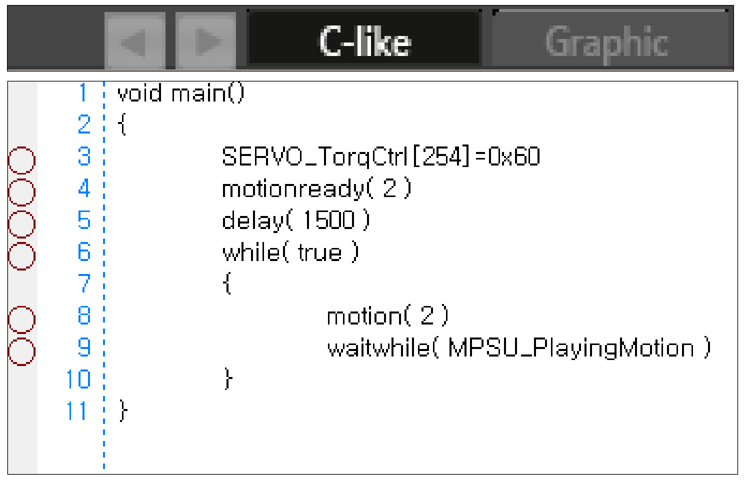
03 Place Module

Drag and dock the module to the Start Point to activate the module. Active module will turn to color.



04 Entire Program

Program loads the saved motion and repeats it continuously.

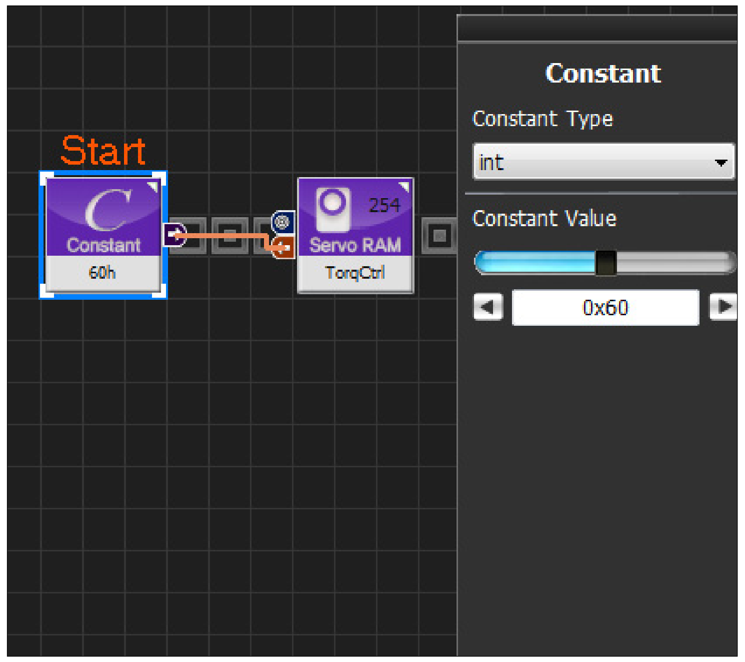


```
void main()
{
        SERVO_TorqCtrl[254]=0x60
        motionready( 2 )
        delay( 1500 )
        while( true )
        {
                motion( 2 )
                waitwhile( MPSU_PlayingMotion )
        }
}
```

05 Viewing C-Like

Click "C-Like" tab near the top right to open the program source code window as shown in the left.

Since codes are very similar to the C language structure, studying the codes will help the user become familiar with the C language as well. Clicking on the module results in cursor jumping to the matching source code line in the C-Like window. This feature enables the user to easily see the codes generated by the module.



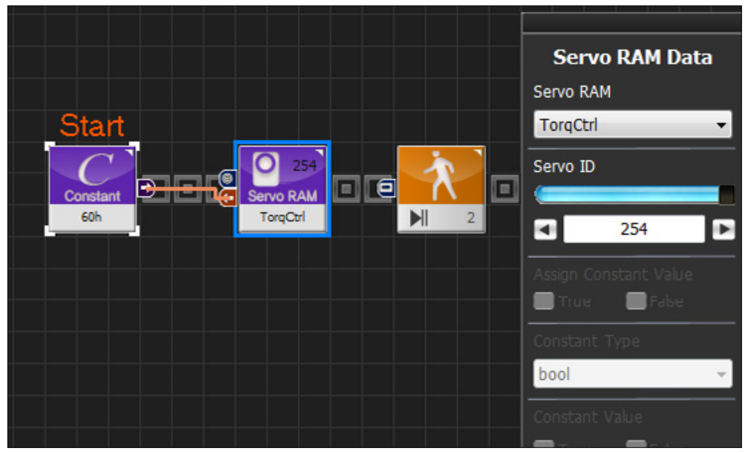
Constant

Constant Type: int

Constant Value: 0x60

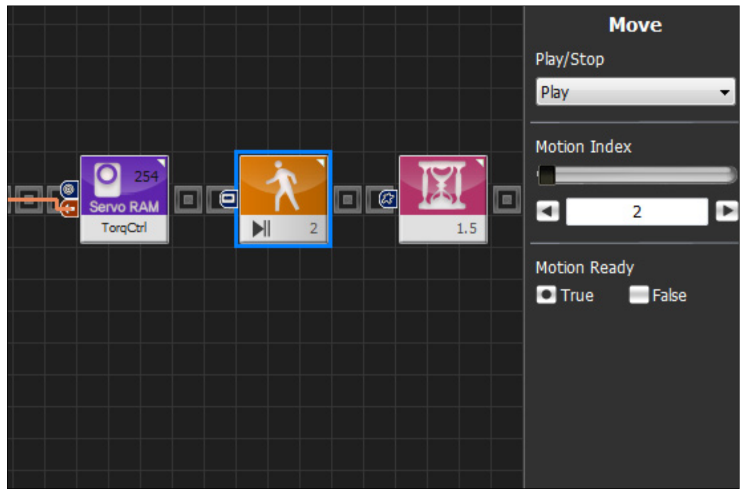
06 Variable Setup

This section allows the servo motor to operate on its own. From the Constant module properties, click on the Constant Value and change the value to 0x60. Constant with Hex number of 0x60 shows the torque enabled state of the servo motor. This value is sent to the input pin of the next module through the output pin.



07 Apply to All Servos

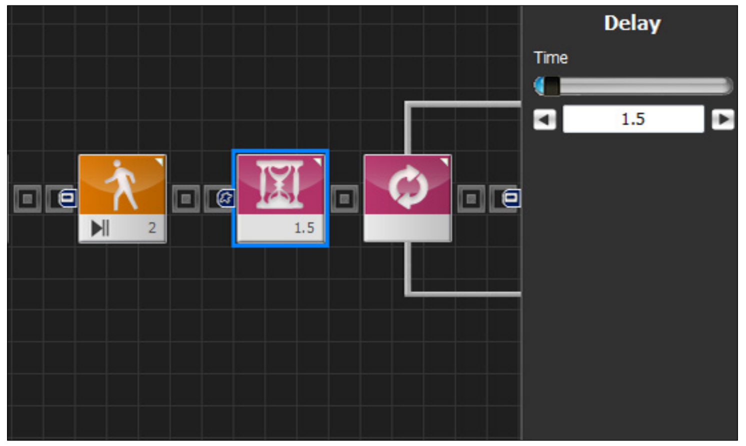
This section applies constant value 0x60 received from the previous section to all servo motors.
Select Data > Servo RAM and place it after the Constant module.
Servo RAM : Select TorqCtrl
Servo ID : Set to 254, ID 254 means setup will be applied to all connected servos.
Use the Connector to connect the output pin of the Constant module to the second input pin of the Servo RAM module.



08 Motion Ready

When motion is first loaded, robot may make a sudden movement. If the difference between the current position and the start of the motion is very different, it may cause stress to the motor or pose danger to the user. To prevent such an occurrence, motion is run in 'Motion Ready' mode to provide time for motion to start.

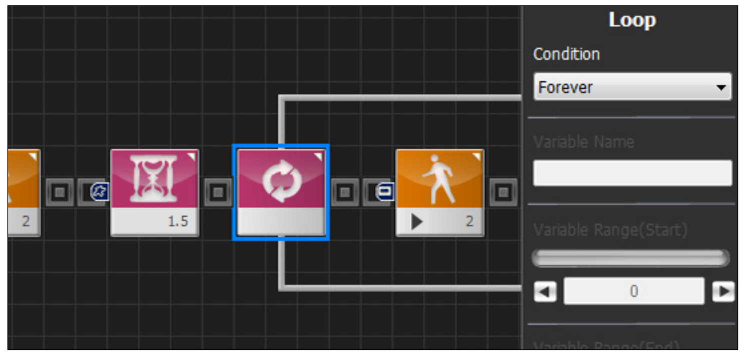
Select Motion > Move module and place it after the previous module.
Play/Stop: Select Play.
Select Motion Index :2, load motion number 2 which makes the robot sit and stand. Since this motion was chosen arbitrarily, user may select another motion number if so desired.
Select Motion Ready: True. When set to True, robot will slowly move to starting position of the motion.



09 Delay

Set Delay value to 1.5s to prevent robot motion from starting before the Motion Ready ends.

Select Flow > Delay module and place it behind the previous module.
Set Time: 1.5. Delay 1.5s.

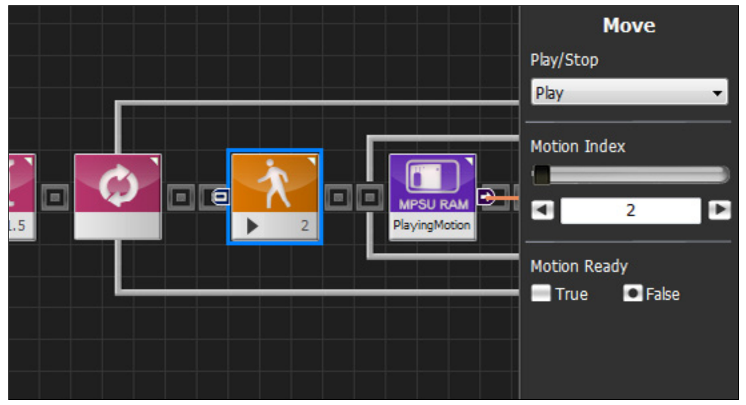


10 Loop

Place loop module and set it to infinite loop mode to continuously repeat the motion. Content of the loop module will be repeated continuously.

Select Flow > Loop and place it after the previous module.

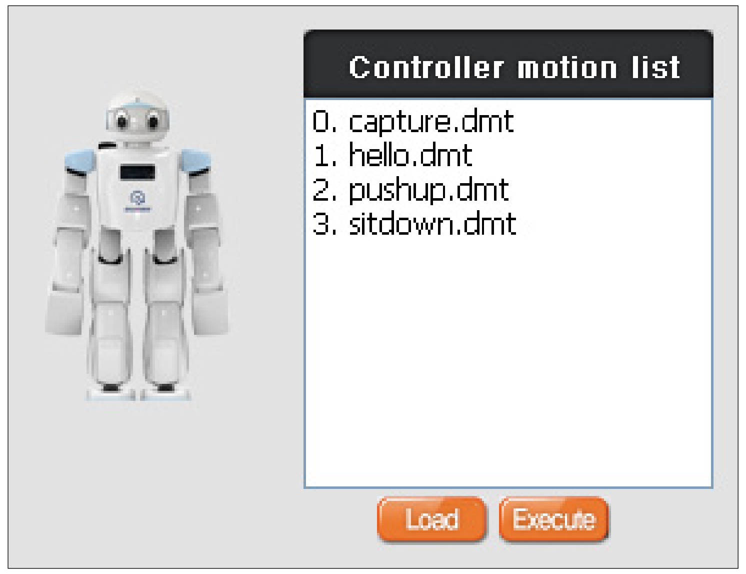
Set Condition : Forever, for infinite loop.



11 Motion Movement

Set the Motion Ready value of the Move module to False to run the motion from start to finish. Run motion number 2.

Select Motion > Move module and place it after the previous module.
Set Play/Stop : Play
Select Motion Index : 2, load motion number 2.
Set Motion Ready : False, run the entire motion.

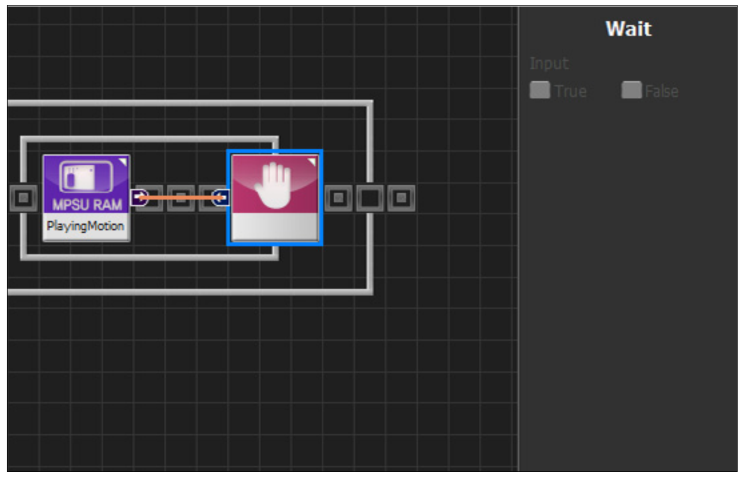


Controller motion list

0. capture.dmt
1. hello.dmt
2. pushup.dmt
3. sitdown.dmt

Reference: View Motion

To view the list of motions in the controller, connect to the robot and click robot setting from DR-SIM.
No 2 motion Robot sits and stands.

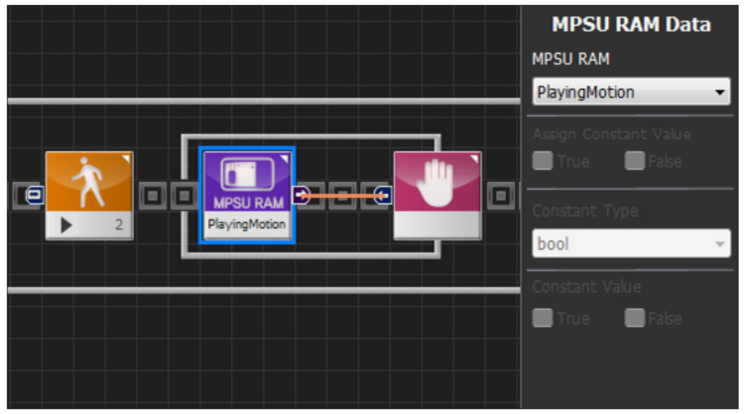


12 Motion Movement Delay

Move module will start the motion and immediately go on to the next module. If loop contains only single move module, move module in the loop will issue a motion command even while the previous motion is still running. This results in difference between the number of motion commands given and actual number of motions played. In order to correct this discrepancy, delay should be introduced to delay the loop from going back to the beginning until the current motion ends.

Variable "PlayingMotion" is found in MPSU RAM. "PlayingMotion" checks the motion to see if motion is in progress. Variable will have value of 1 if motion is in progress and 0 if motion is not active. Adding Delay with PlayMotion as the condition will delay the loop until the current motion ends.

Select Flow > Wait module and place it after the previous module.



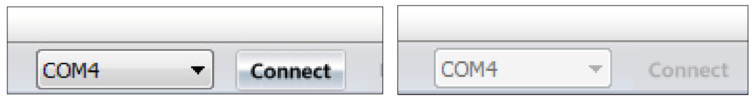
MPSU RAM Data

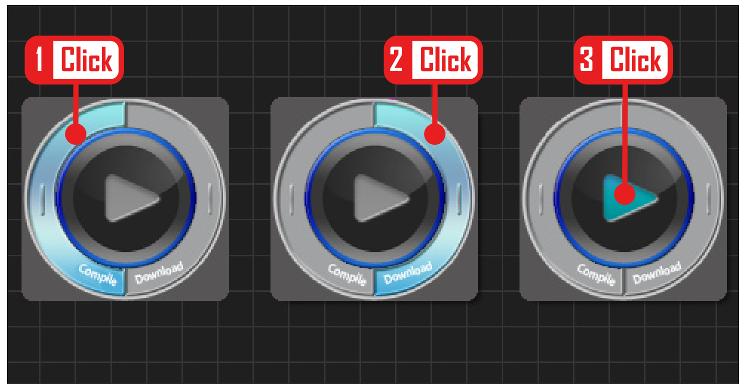
MPSU RAM: PlayingMotion

13 Connecting Motion Movement Variable

Create a routine to wait for motion to end with PlayingMotion as the condition.

Select Data > MPSU RAM module and place it inside the left outline of the Wait module.
Select MPSU RAM : PlayingMotion
Connect the MPSU RAM module output pin to the input pin of the Wait module. This will make the Wait module delay the program as long as PlayingMotion value is true (Not 0). When the motion ends, delay will end and program will go back to the beginning of the loop to run the motion again.

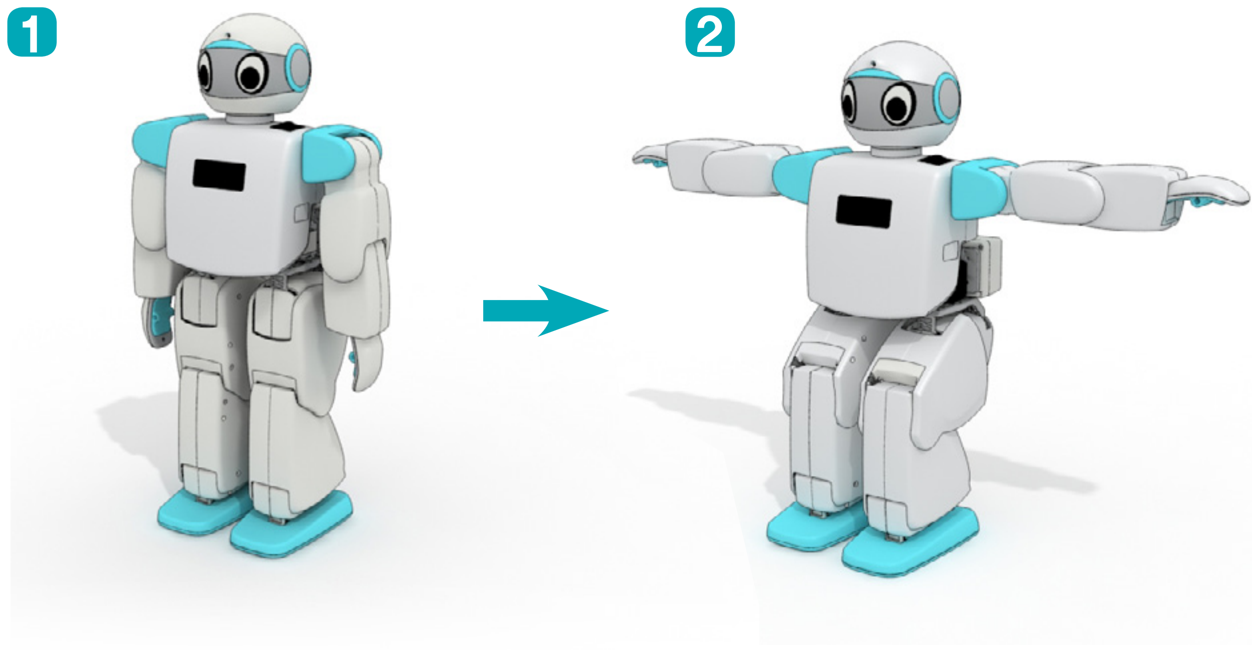




14 Download

After programming is completed.
Compile -> Download -> Run.

Select the appropriate COMPORT or USB-to-Serial number located on the PC and press Connect to open the serial port. Click 'Compile' and then click 'Download' on the right to download the program to the robot if no compile error is found. Click "Run" button(arrow in the middle)after completing the download.



15 Robot Motion

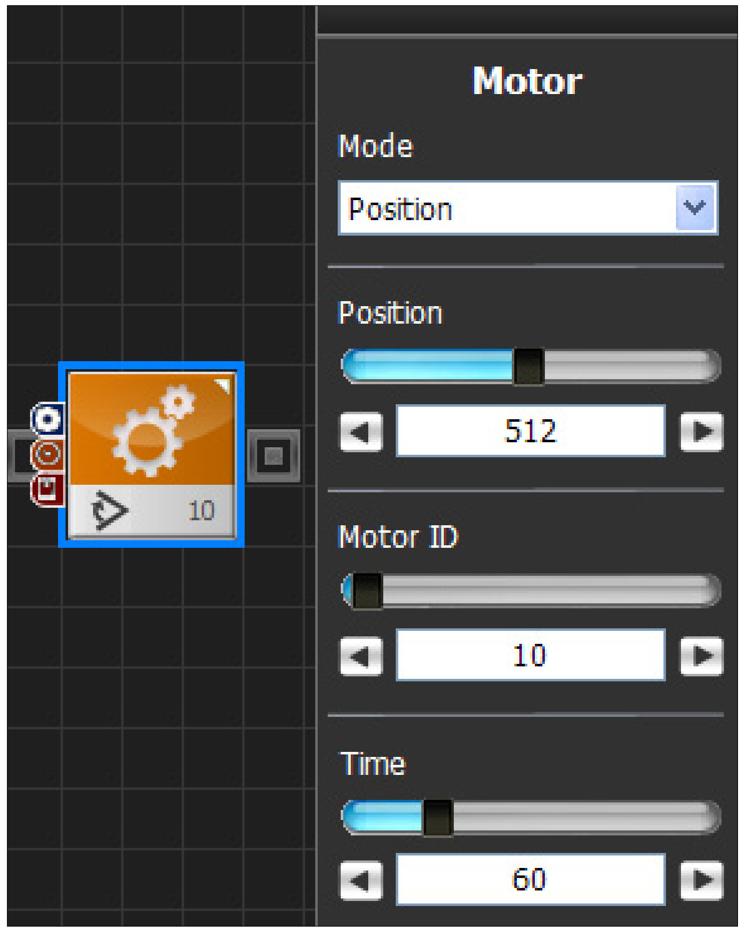
Robot wiil continuously repeat sit and stand motion.

Programming Individual Module

Motor 08-2

Motor Description

Motor module has two types of oerating modes. Positions control mode and Speed control model.



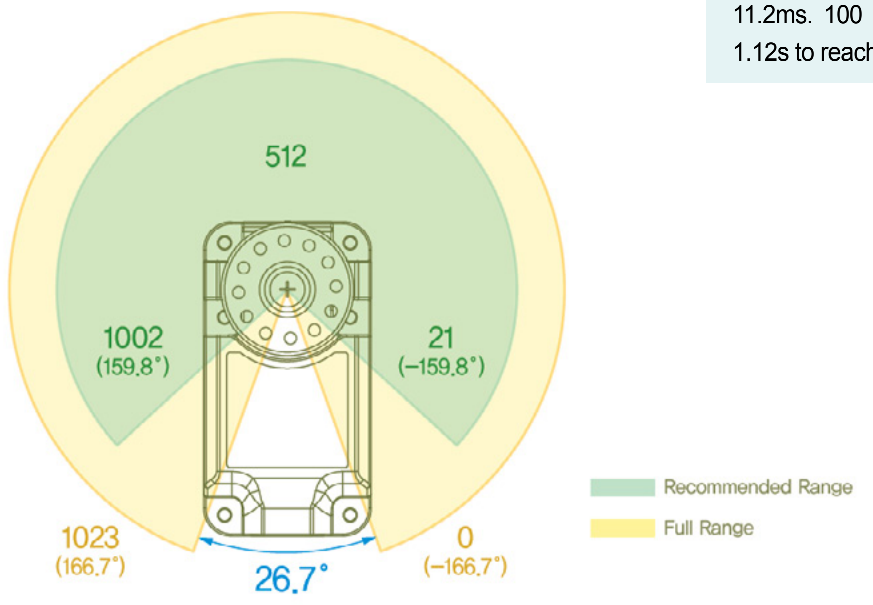
1 Position Control Mode

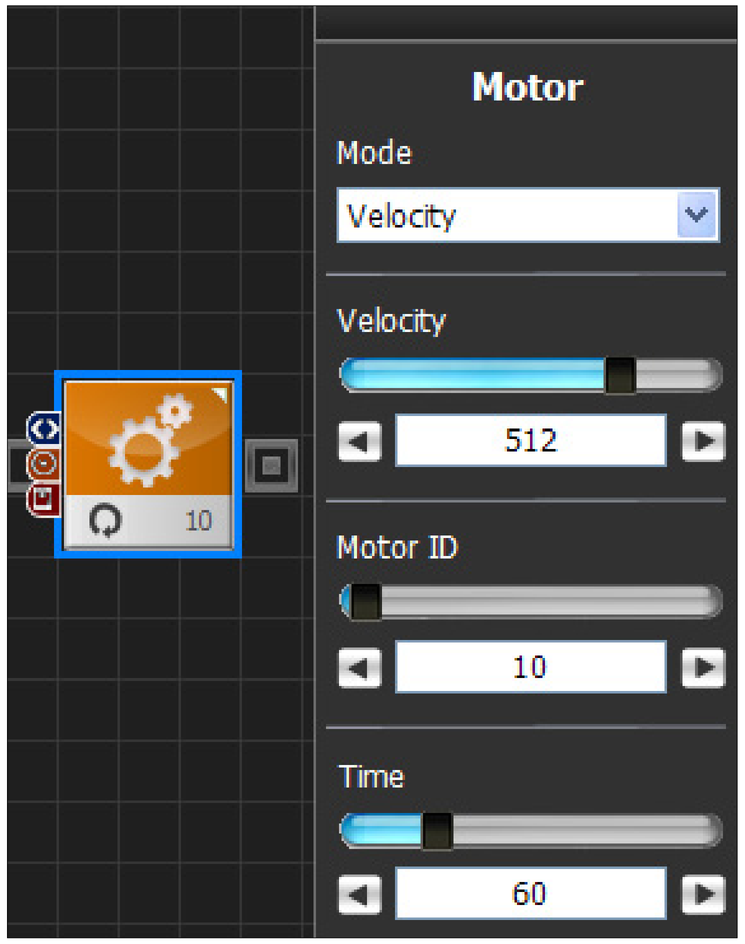
Position control mode changes the position of the selected motor to desired position.

Position has value range between -127 ~ 1152. Servos are released from the factory with default value range of 21 ~ 1002. Values beyond the default range is possible with adjustment to the min/max motor values and position adjustment. Motor has regular position value of 512 which is used as a standard position value when assembling. When all Hovis motors have position value of 512, Hovis will be in standing position with both arms stretched out 90 degrees to the side. Refer to the diagram below to view position range and regular position.

Motor ID is the ID of the servo to be controlled.

Time refers to the time it takes for servo to reache the goal position. 1tick = 11.2ms. 100 tick would take the servo 1.12s to reach the goal position.





2 Speed(Velocity) Control Mode

Speed control mode puts the selected servo in continous roation with specific speed.

Velocity has value range of -1023~1023, Larger the value, larger the output with increased rotation speed. Sign of the value determines the direction of the rotation.

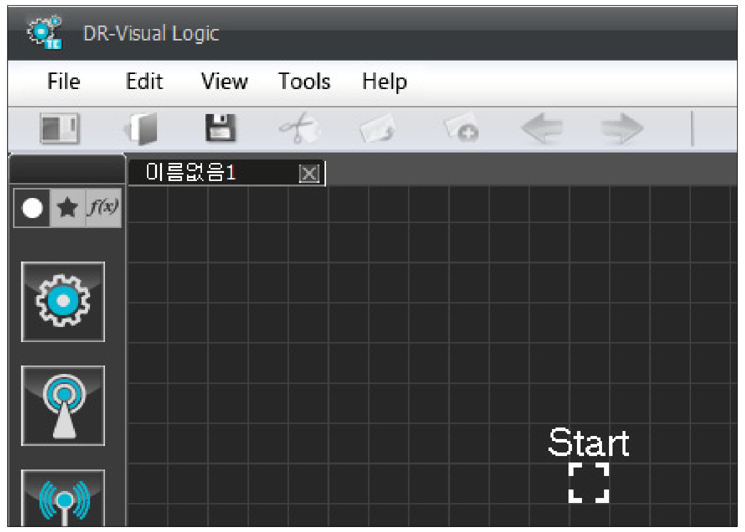
Motor ID is the ID of the servo to be controlled.

Time refers to the time it takes for servo to reache the goal position. 1tick = 11.2ms. When set to 100 tick, servo would take 1.12s to gradually reach the goal speed.

Example Description

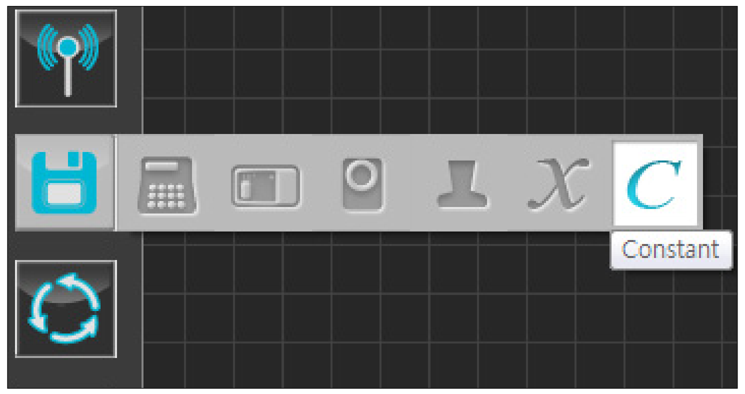
Robot motions are usually made by controlling and consolidating each individual servo movements. But, due to the complexity of controlling each servo, motions are usually made by tools such as DR-SIM. However, in this example program, instead of using DR-SIM, DR-Visual Logic will be used to control each individual servo to produce continuous motion. The end result of the program will a very interesting wave dancing robot motion.

※ Caution - The arm structure of HOVIS Eco is not suitable for this wave dance example because of its limitation of movement range. You can only reference this example to make a your own motion.



01 Create New File

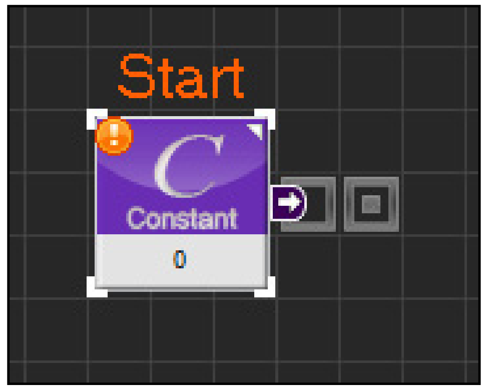
Click on the left most icon in the tools to create new dts file. Select HOVIS Lite/Eco as the Robot.



02 Select Module

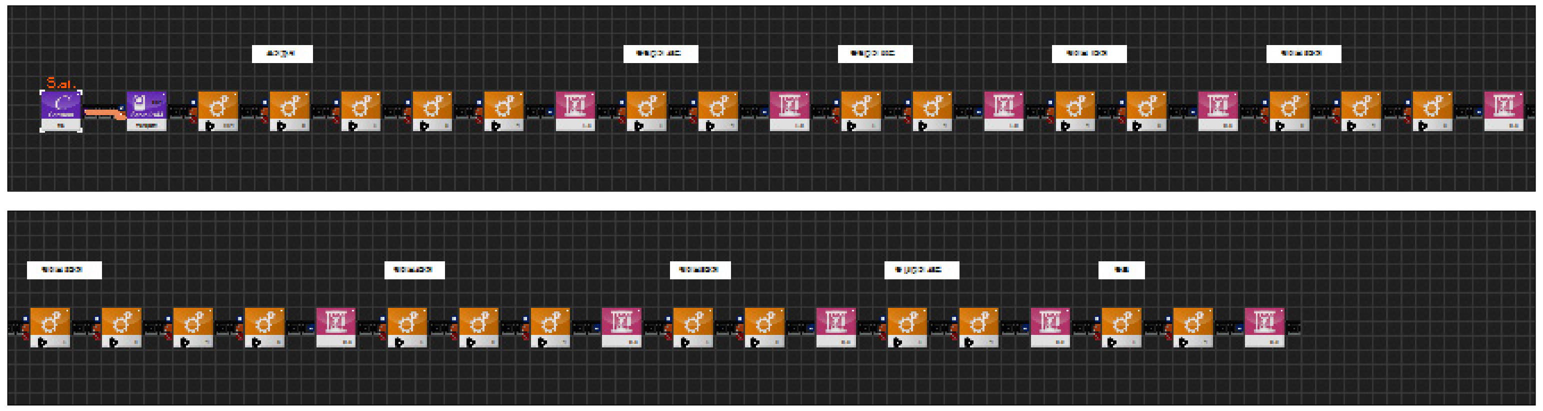
Prior to placing the module, click to select the module from the module bar located on the left side.

Click Data > Constant module



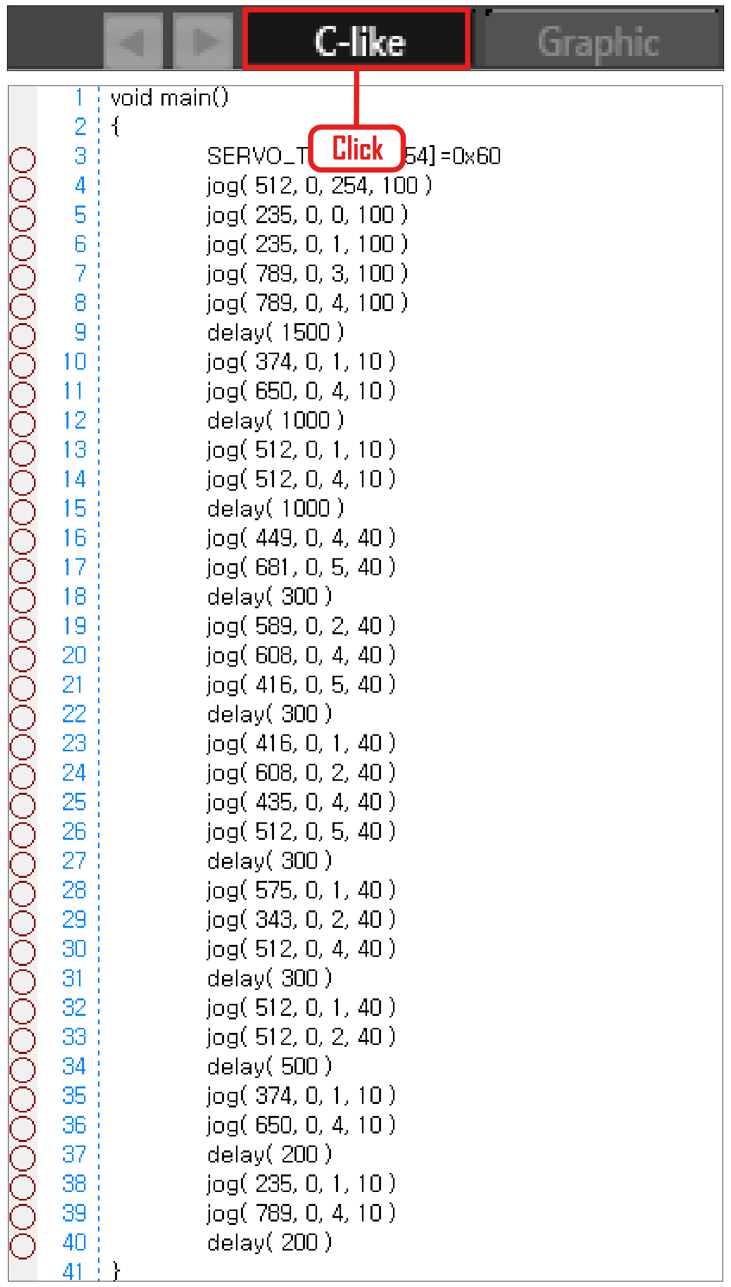
03 Place Module

Drag and dock the module to the Start Point to activate the module. Active module will turn to color.



04 Entire Program

View of the entire program.

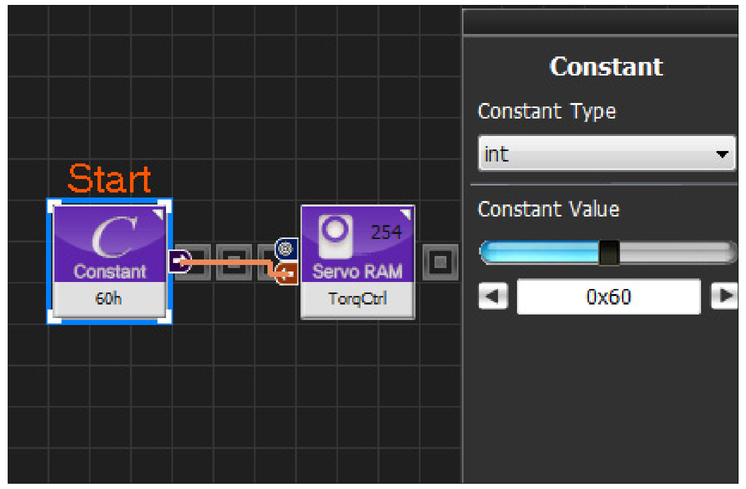


```
void main()
{
        SERVO_T[illegible]54] =0x60
        jog( 512, 0, 254, 100 )
        jog( 235, 0, 0, 100 )
        jog( 235, 0, 1, 100 )
        jog( 789, 0, 3, 100 )
        jog( 789, 0, 4, 100 )
        delay( 1500 )
        jog( 374, 0, 1, 10 )
        jog( 650, 0, 4, 10 )
        delay( 1000 )
        jog( 512, 0, 1, 10 )
        jog( 512, 0, 4, 10 )
        delay( 1000 )
        jog( 449, 0, 4, 40 )
        jog( 681, 0, 5, 40 )
        delay( 300 )
        jog( 589, 0, 2, 40 )
        jog( 608, 0, 4, 40 )
        jog( 416, 0, 5, 40 )
        delay( 300 )
        jog( 416, 0, 1, 40 )
        jog( 608, 0, 2, 40 )
        jog( 435, 0, 4, 40 )
        jog( 512, 0, 5, 40 )
        delay( 300 )
        jog( 575, 0, 1, 40 )
        jog( 343, 0, 2, 40 )
        jog( 512, 0, 4, 40 )
        delay( 300 )
        jog( 512, 0, 1, 40 )
        jog( 512, 0, 2, 40 )
        delay( 500 )
        jog( 374, 0, 1, 10 )
        jog( 650, 0, 4, 10 )
        delay( 200 )
        jog( 235, 0, 1, 10 )
        jog( 789, 0, 4, 10 )
        delay( 200 )
}
```

05 Viewing C-Like

Click "C-Like" tab near the top right to open the program source code window as shown in the left.

Since codes are very similar to the C language structure, studying the codes will help the user become familiar with the C language as well. Clicking on the module results in cursor jumping to the matching source code line in the C-Like window. This feature enables the user to easily see the codes generated by the module.


Start

Constant
60h

Servo RAM
254
TorqCtrl

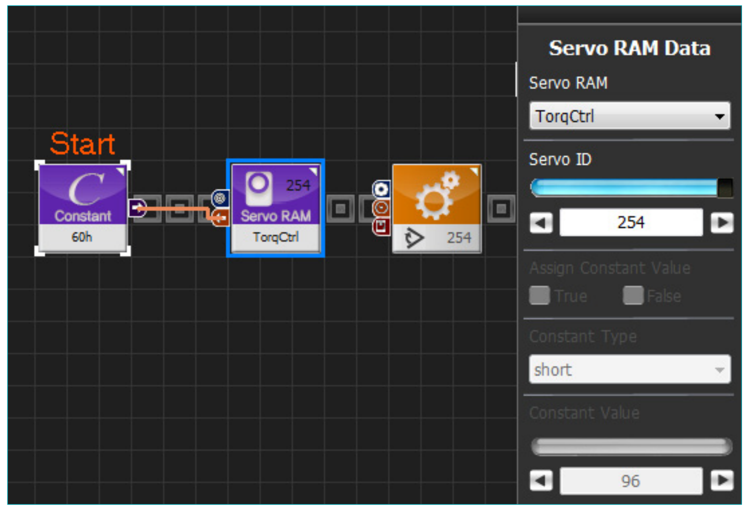
Constant

Constant Type: int

Constant Value: 0x60

06 Set Constant

This section allows the servo motor to operate on its own. From the Constant module properties, click on the Constant Value and change the value to 0x60. Constant with Hex number of 0x60 shows the torque enabled state of the servo motor. This value is sent to the input pin of the next module through the output pin.


Start

Constant
60h

Servo RAM
254
TorqCtrl

254

Servo RAM Data

Servo RAM: TorqCtrl

Servo ID: 254

Assign Constant Value: True False

Constant Type: short

Constant Value: 96

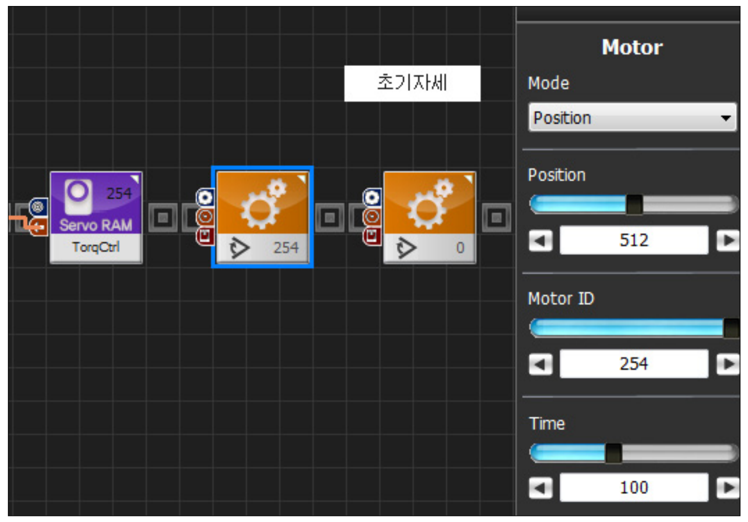
07 Apply to All servos

This section applies constant value 0x60 received from the previous section to all servo motors.

Select Data > Servo RAM and place it after the Constant module.

Servo RAM : Select TorqCtrl
Servo ID : Set to 254, ID 254 means setup will be applied to all connected servos.

Use the Connector to connect the output pin of the Constant module to the second input pin of the Servo RAM module.


초기자세

254
Servo RAM
TorqCtrl

254

0

Motor

Mode: Position

Position: 512

Motor ID: 254

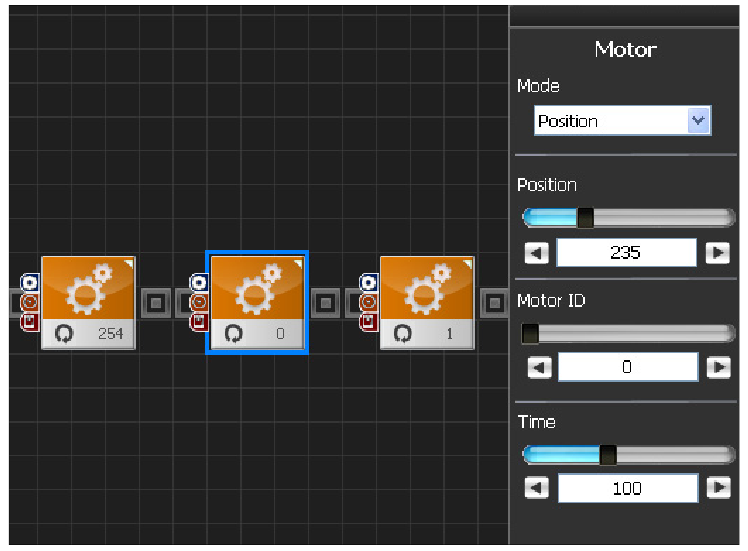
Time: 100

08 Set Position to All Servos

This section sets all servo motor positions to the center.

Select Motion > Motor and place it after the previous module.

Set Mode : Position, to control the angle.
Set Position : 512, Position value of 512 will send the motor to the 0 point (center).
Set Motor ID : 254, ID 254 will apply the setup to all connected servo motors.
Set Time : 100, 1 unit = 11.2ms, value of 100 is approximately 1.12s. Motors will be sent to the desired position in 1.12s.


254

0

1

Motor

Mode: Position

Position: 235

Motor ID: 0

Time: 100

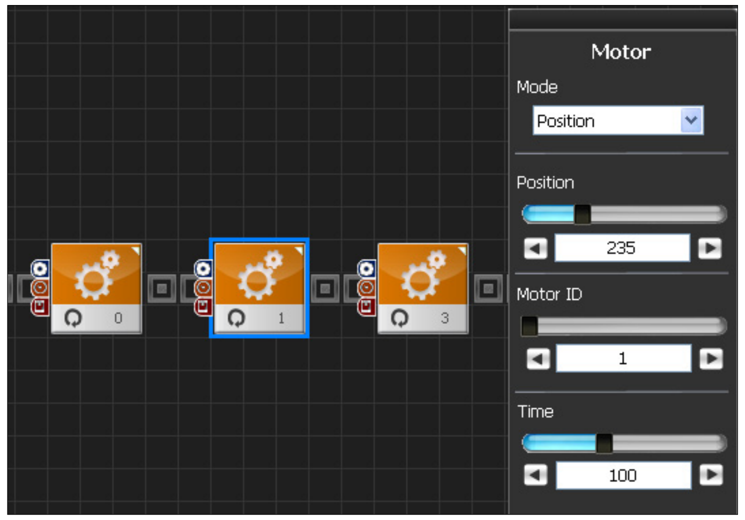
09 Moto ID 0 (Right Shoulder) Setup

1st Stage: Initial Position

When all servo motors are aligned to the center, humanoid robot will be standing with both arms stretched out to the side. In order to make applying motion easier, robot posture should be returned to the basic attention posture with both arms lowered to the side of the body.

Select Motion > Motor and place it after the previous module.

Set Mode : Position, to control the angle.
Set Position : 235. Changing the motor position to 235 will allow the horizontally stretched arm to be lowered to vertical position.
Set Motor ID : 0, right shoulder motor has ID 0.
Set Time : 100, motor will move to the required position in approximately 1.12s.

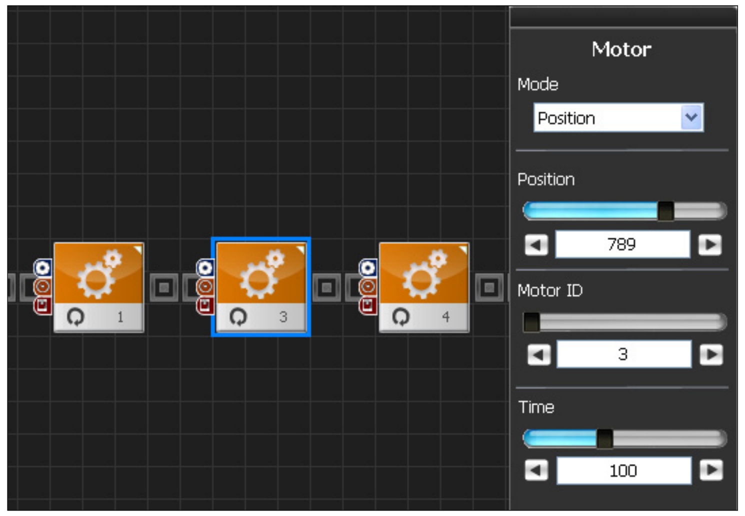


Motor
Mode: Position
Position: 235
Motor ID: 1
Time: 100

10 Set Motor ID 1 (Upper Right Arm)

Select Motion > Motor module and place it after the previous module.

Set Mode : Position to control the motor position.
Set Position : 235. Changing the motor position to 235 will lower the horizontally stretched arm to the new vertical position.
Set Motor ID : 1, upper right arm motor has ID 1.
Set Time : 100, motor will move to the required position in approximately 1.12s.

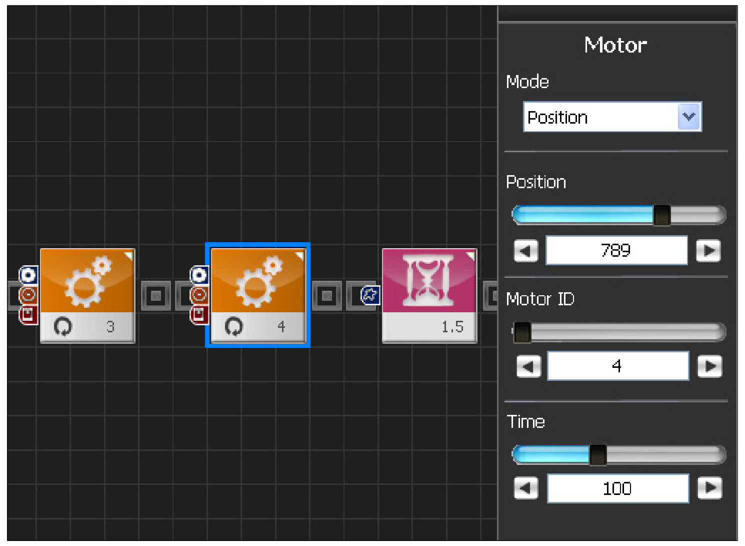


Motor
Mode: Position
Position: 789
Motor ID: 3
Time: 100

11 Set Motor ID 3 (Left Shoulder)

This section turns the left shoulder in order to lower the arm to the side of the body.

Select Motion > Motor module and place it after the previous module
Set Mode : Position to control the motor position.
Set Position : 789. Changing the motor position to 789 will allow the horizontally stretched arm to be lowered to vertical position.
Set Motor ID : 3 , left shoulder motor has ID 0.
Set Time : 100, motor will move to the required position in approximately 1.12s.

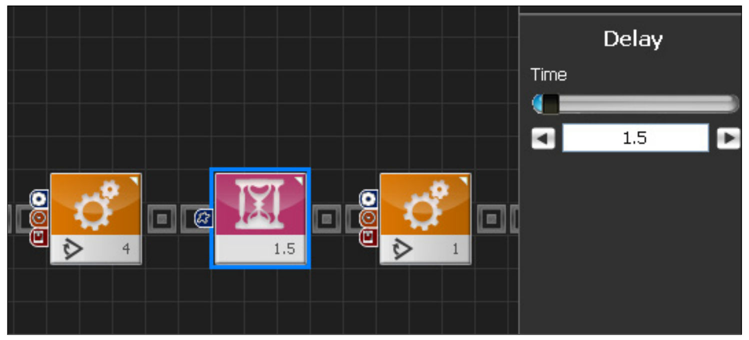


Motor
Mode: Position
Position: 789
Motor ID: 4
Time: 100

12 Set Motor ID 4 (Upper Left Arm)

Select Motion > Motor module and place it after the previous module.

Set Mode : Position to control the motor position.
Set Position : 789. Changing the motor position to 789 will lower the horizontally stretched arm to the new vertical position.
Set Motor ID : 4, upper left arm motor has ID 4.
Set Time : 100, motor will move to the required position in approximately 1.12s.

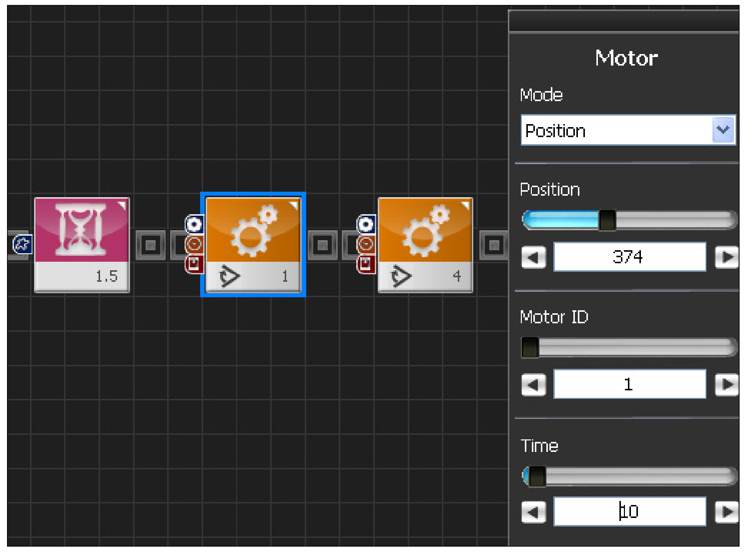


Delay
Time: 1.5

13 Delay

This section adds 1.5s of delay before starting the next motor operation. Delay allows the motor to move to the position requested by the previous module without any interruption. Individual motion command to the motors 0 1 3 4 were sent immediately after the command to send all motors (254) to position 512. From users perspective, it would seem like robot lowered both arms simultaneously to take attention posture.

Select Flow > Delay module and place it after the previous module.
Set Time : 1.5s, delay 1.5s.

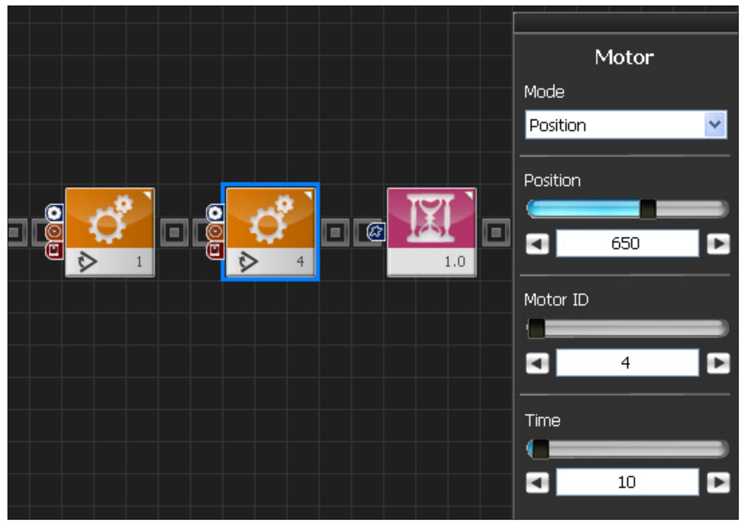


Motor
Mode: Position
Position: 374
Motor ID: 1
Time: 10

14 Set Motor ID1 (Upper Right Arm)

2nd Stage : Change right arm angle to 45 degrees to prepare for dance motion.

Select Motion > Motor
Set Mode : Position
Set Position : 374,
Position : value 374 will change the angle of the upper right arm to 45 degrees.
Set Motor ID : 1, upper right arm motor has ID 1.
Set Time : 10, Motor will move to required position in 0.112s.

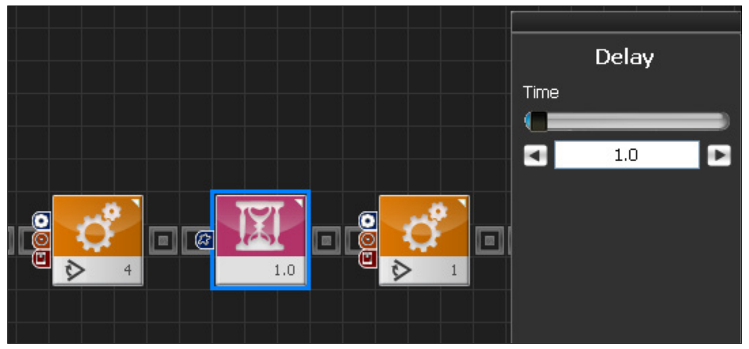


Motor
Mode: Position
Position: 650
Motor ID: 4
Time: 10

15 Set Motor ID4 (Upper Left Arm)

Changes upper left arm motor angle to 45 degrees

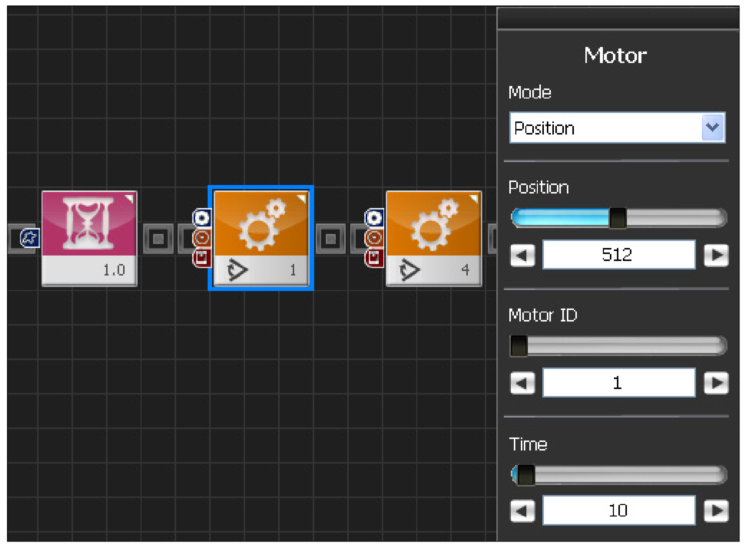
Select Motion > Motor
Set Mode : Position
Set Position : 650,
Position value 650 will change the angle of the upper left arm to 45 degrees.
Set Motor ID : 4, upper left arm motor has ID 4.
Set Time : 10, Motor will move to required position in 0.112s.


Delay
Time
1.0

16 Delay

Delays for 1.0s before starting next motion.

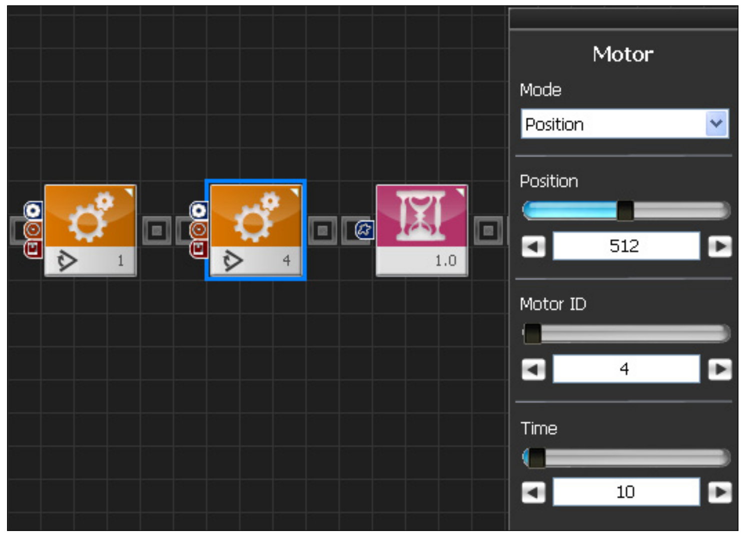
Select Flow > Delay
Set Time : 1.0


Motor
Mode
Position
Position
512
Motor ID
1
Time
10

17 Set Motor ID1 (Upper Right Arm)

3rd Stage : Change right arm angle to 90 degrees to start wave dance motion.

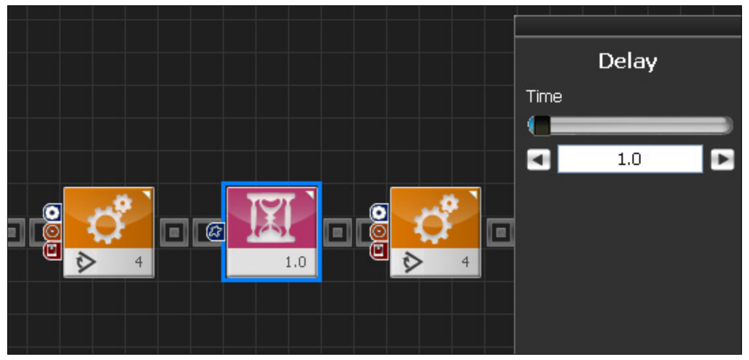
Select Motion > Motor
Set Mode : Position
Set Position : 512,
Position value 512 (Center position) will change the angle of the upper right arm to 90 degrees.
Set Motor ID : 1, upper right arm motor has ID 1.
Set Time : 10, Motor will move to required position in 0.112s.


Motor
Mode
Position
Position
512
Motor ID
4
Time
10

18 Set Motor ID4 (Upper Left Arm)

Changes upper left arm motor angle to 90 degrees.

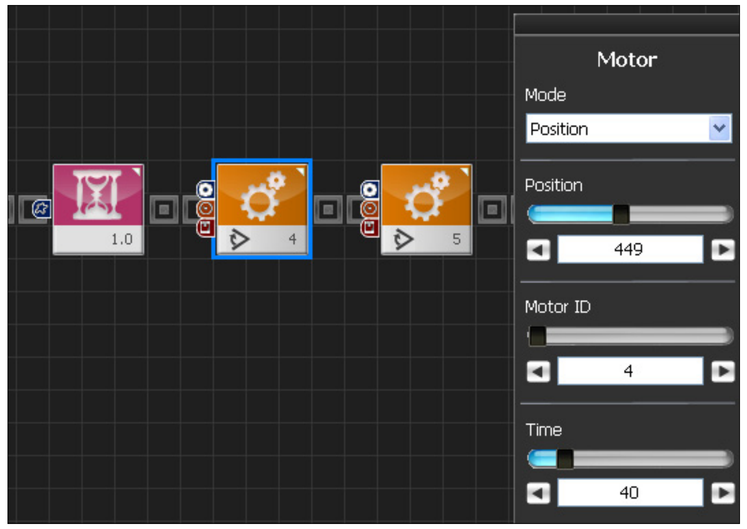
Select Motion > Motor
Set Mode : Position
Set Position : 512,
Position value 512 will change the angle of the upper left arm to 90 degrees.
Set Motor ID : 4, upper left arm motor has ID 4.
Set Time : 10, Motor will move to required position in 0.112s.


Delay
Time
1.0

19 Delay

Delays for 1.0s before starting next motion.

Select Flow > Delay
Set Time : 1.0

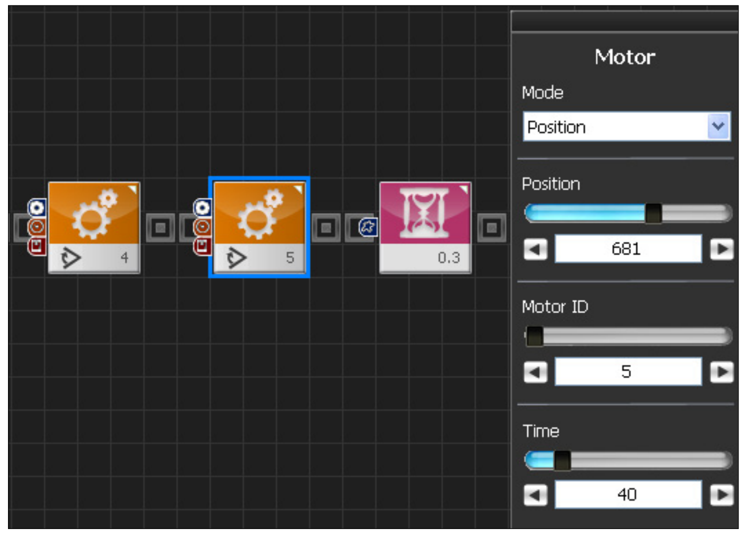

Motor
Mode
Position
Position
449
Motor ID
4
Time
40

20 Set Motor ID4 (Upper Left Arm)

4th Stage : Wave 1st step

Start wave from the left arm.

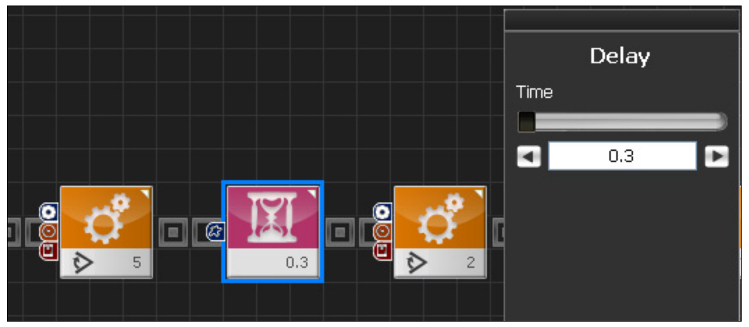
Select Motion > Motor
Set Mode : Position
Set Position : 449
Set Motor ID : 4
Set Time : 40, Motor will move to required position in 0.448


Motor
Mode
Position
Position
681
Motor ID
5
Time
40

21 Set Motor ID5(Lower Left Arm)

Control lower left arm motor ID5.

Select Motion > Motor
Set Mode : Position
Set Position : 681
Set Motor ID : 5
Set Time : 40, Motor will move to required position in 0.448

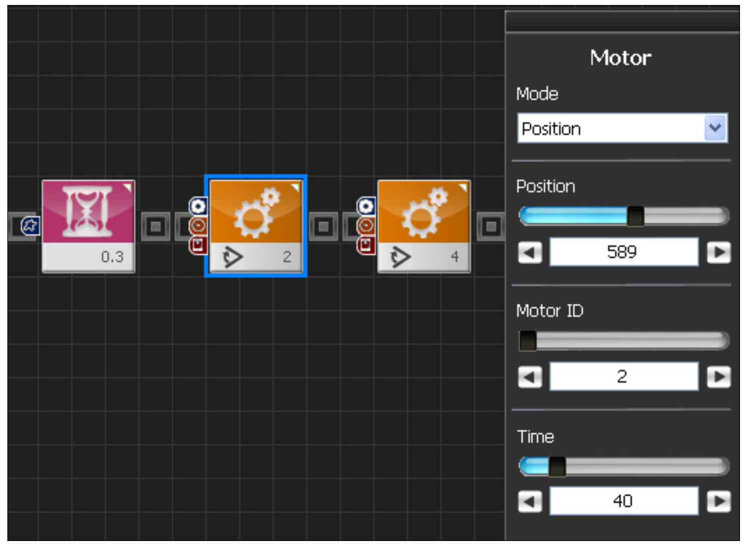


Delay
Time
0.3

22 Delay

Delay 0.3s before starting next motion. Delay time should be less than the motor motion time for smooth motion.

Select Flow > Delay



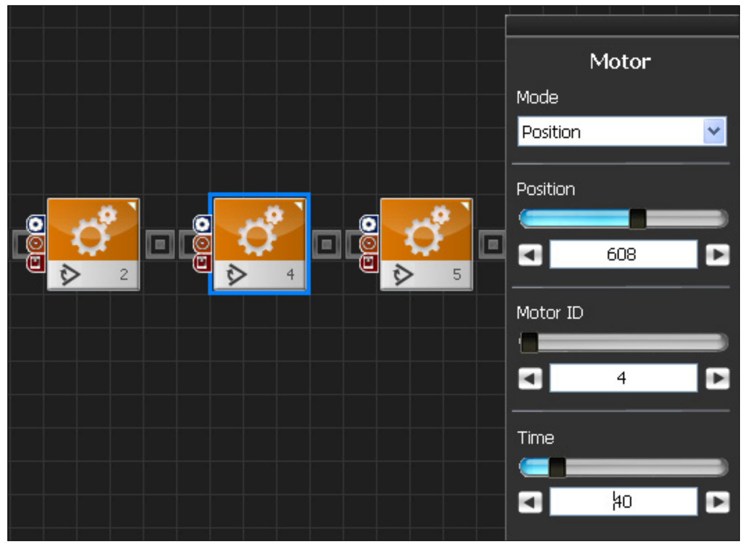
Motor
Mode: Position
Position: 589
Motor ID: 2
Time: 40

23 Set Motor ID2 (Lower Right Arm)

5th Stage : Wave 2nd step

Start right arm wave from motor ID2.

Select Motion > Motor
Set Mode : Position
Set Position : 589
Set Motor ID : 2
Set Time : 40, Motor will move to required position in 0.448s

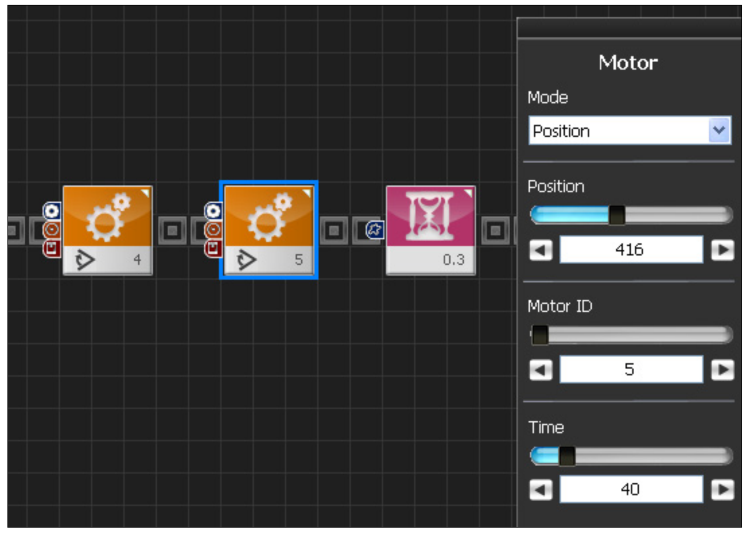


Motor
Mode: Position
Position: 608
Motor ID: 4
Time: 40

24 Set Motor ID4 (Upper Left Arm)

Change the motor angle slightly to give wave effect.

Select Motion > Motor
Set Mode : Position
Set Position : 608
Set Motor ID : 4
Set Time : 40, Motor will move to required position in 0.448s

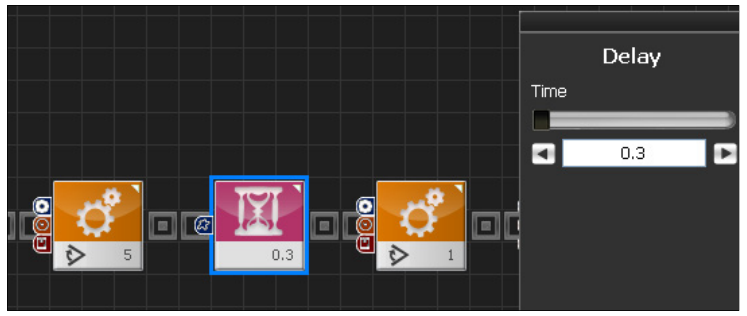


Motor
Mode: Position
Position: 416
Motor ID: 5
Time: 40

25 Set Motor ID5 (Lower Left Arm)

Control lower left arm motor ID5

Select Motion > Motor
Set Mode : Position
Set Position : 416
Set Motor ID : 5
Set Time : 40, Motor will move to required position in 0.448s

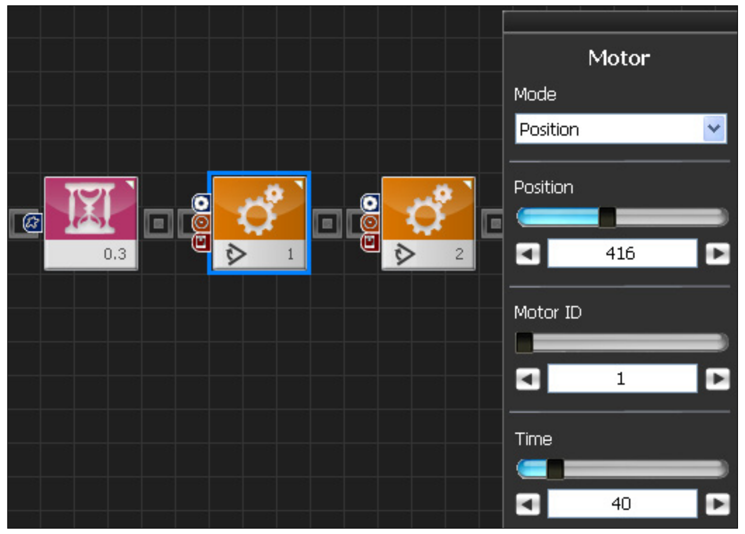


Delay
Time
0.3

26 Delay

Delay 0.3s before starting next motion. Delay time should be less than the motor motion time for smooth motion.

Select Flow > Delay
Set Time : 0.3



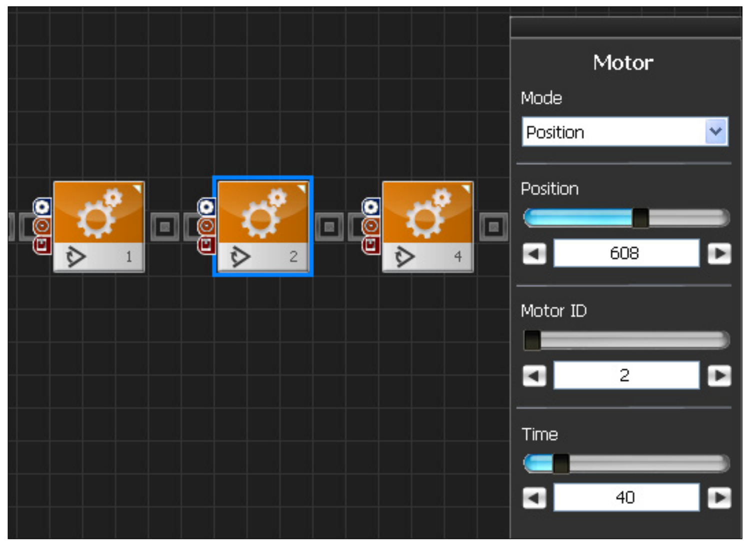
Motor
Mode: Position
Position: 416
Motor ID: 1
Time: 40

27 Set Motor ID1 (Upper Right Arm)

6th Stage : Wave 3nd step

Wave reaches to upper right arm.
Start right arm wave from motor ID2.

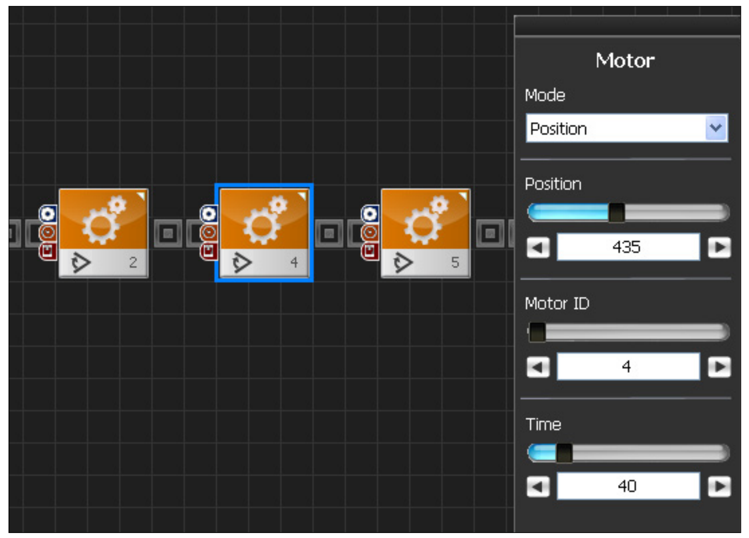
Select Motion > Motor
Set Mode : Position
Set Position : 416
Set Motor ID : 1
Set Time : 40, Motor will move to required position in 0.448s


Motor
Mode: Position
Position: 608
Motor ID: 2
Time: 40

28 Set Motor ID2 (Lower Right Arm)

Control lower right arm motor ID2.

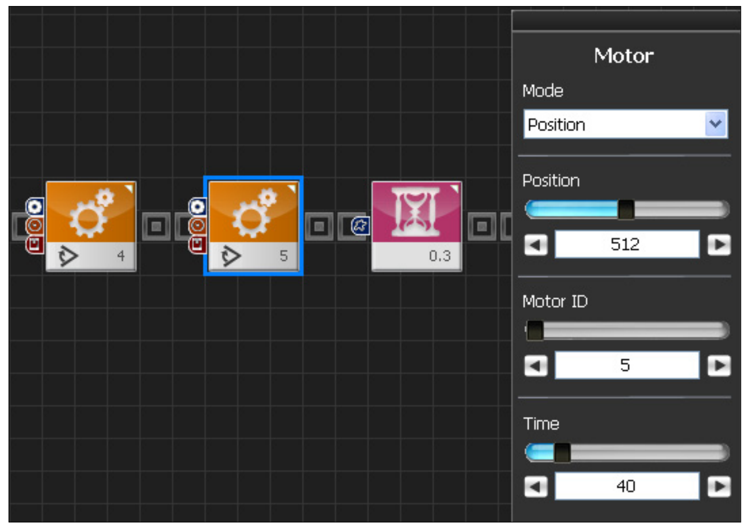
Select Motion > Motor
Set Mode : Position
Set Position : 608
Set Motor ID : 2
Set Time : 40, Motor will move to required position in 0.448s


Motor
Mode: Position
Position: 435
Motor ID: 4
Time: 40

29 Set Motor ID4 (Upper Left Arm)

Control upper left arm motor ID4.

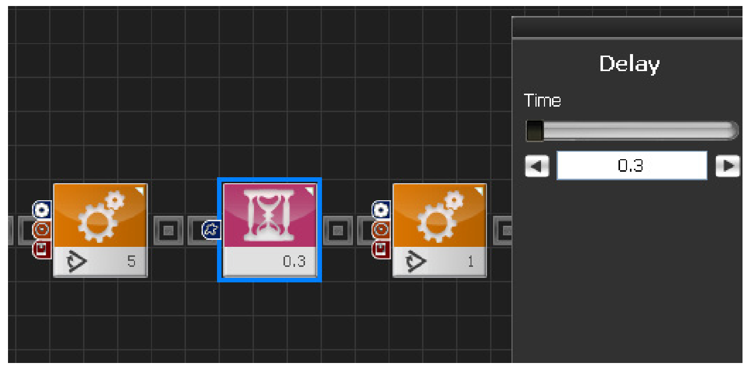
Select Motion > Motor
Set Mode : Position
Set Position : 435
Set Motor ID : 4
Set Time : 40, Motor will move to required position in 0.448s


Motor
Mode: Position
Position: 512
Motor ID: 5
Time: 40

30 Set Motor ID5 (Lower Left Arm)

Return lower left arm motor ID5 to initial position 512.

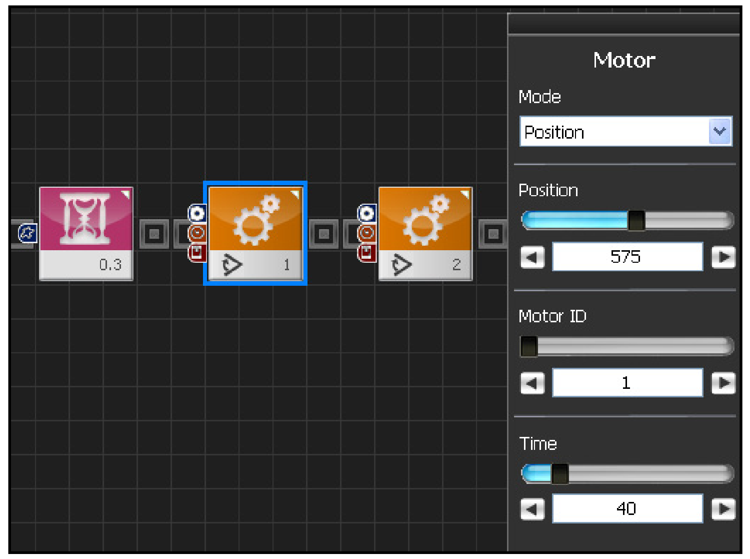
Select Motion > Motor
Set Mode : Position
Set Position : 512
Set Motor ID : 5
Set Time : 40, Motor will move to required position in 0.448s


Delay
Time: 0.3

31 Delay

Delay 0.3s before starting next motion. Delay time should be less than the motor motion time for smooth motion.

Select Flow > Delay
Set Time : 0.3

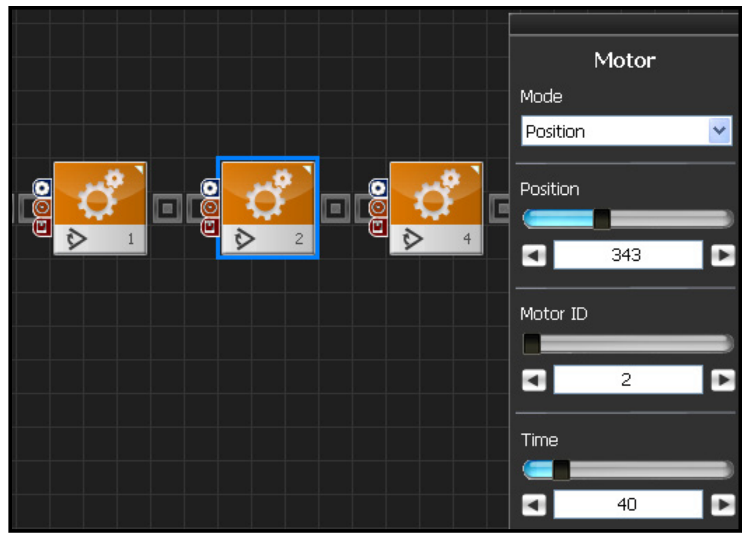

Motor
Mode: Position
Position: 575
Motor ID: 1
Time: 40

32 Set Motor ID1 (Upper Right Arm)

7th stage : Wave 4th step.

End the left arm wave and start last right arm wave

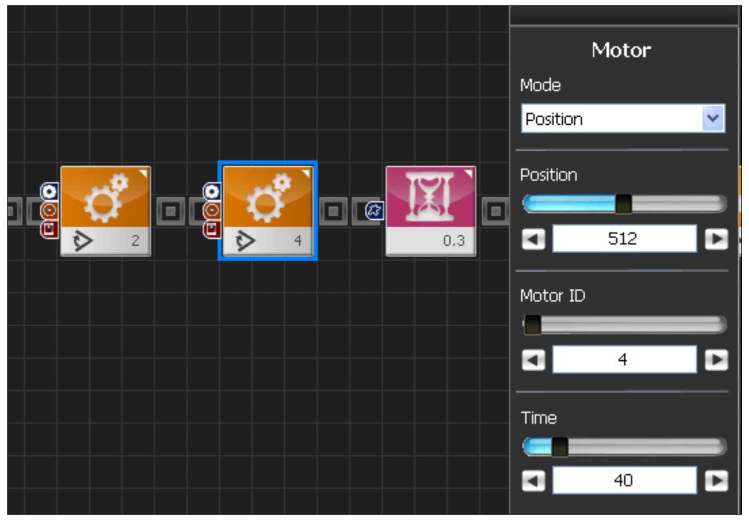
Select Motion > Motor
Set Mode : Position
Set Position : 575
Set Motor ID : 1
Set Time : 40, Motor will move to required position in 0.448s


Motor
Mode: Position
Position: 343
Motor ID: 2
Time: 40

33 Set Motor ID2 (Lower Right Arm)

Control lower right arm ID2.

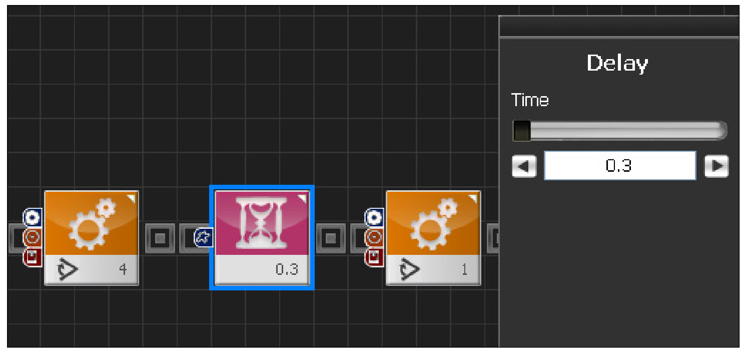
Select Motion > Motor
Set Mode : Position
Set Position : 343
Set Motor ID : 2
Set Time : 40, Motor will move to required position in 0.448s



34 Set Motor ID4 (Upper Left Arm)

Return upper left arm motor ID4 to initial position 512.

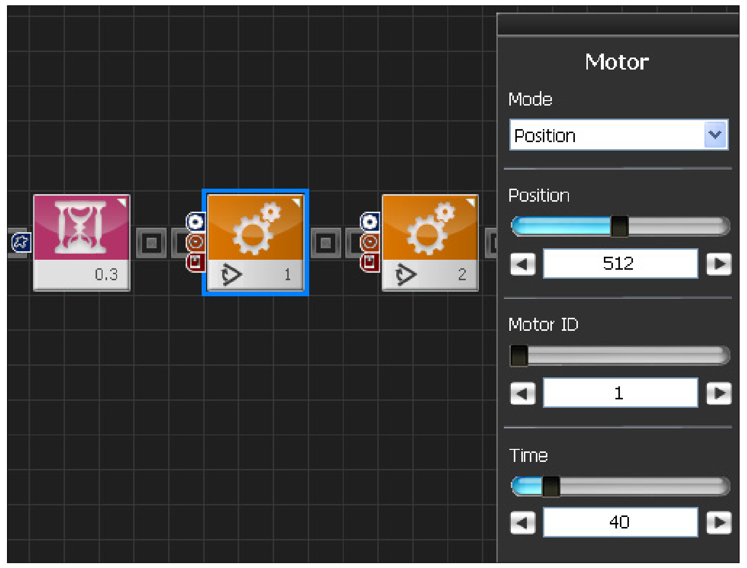
Select Motion > Motor
Set Mode : Position
Set Position : 512
Set Motor ID : 4
Set Time : 40, Motor will move to required position in 0.448s



35 Delay

Delay 0.3s before starting next motion. Delay time should be less than the motor motion time for smooth motion.

Select Flow > Delay
Set Time : 0.3

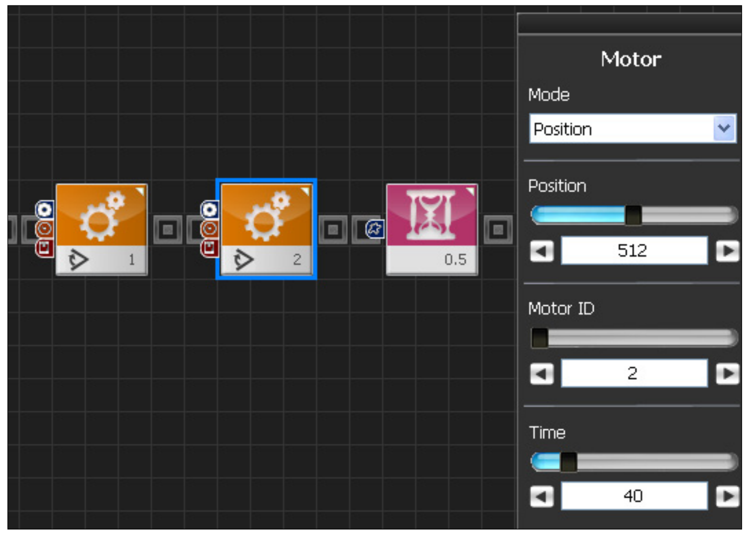


36 Set Motor ID1 (Upper Right Arm)

8th stage : Wave 5th step.

Return the arms to initial stretched position.

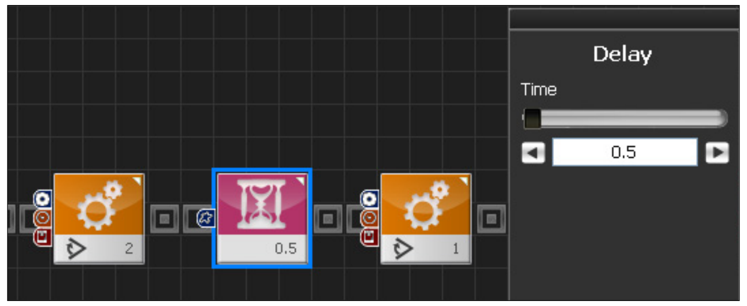
Select Motion > Motor
Set Mode : Position
Set Position : 512
Set Motor ID : 1
Set Time : 40, Motor will move to required position in 0.448s



37 Set Motor ID2 (Lower Right Arm)

Return motor ID2 to initial position 512.

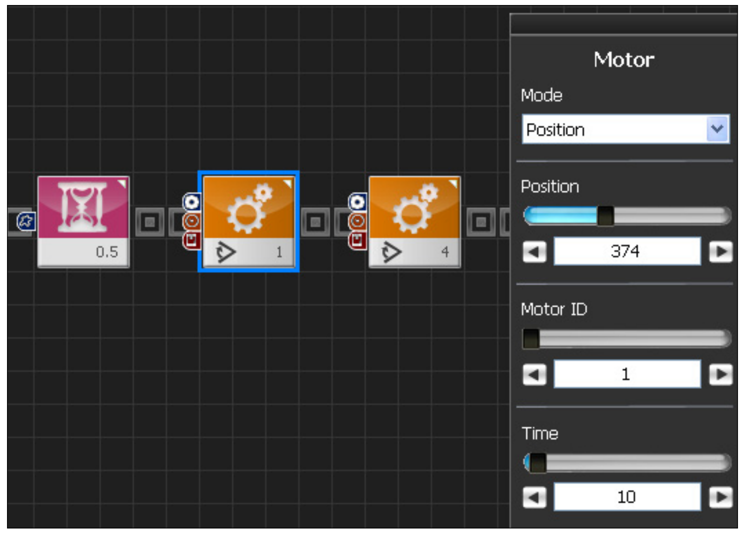
Select Motion > Motor
Set Mode : Position
Set Position : 512
Set Motor ID : 2
Set Time : 40, Motor will move to required position in 0.448s



38 Delay

Delay 0.5s before starting next motion.

Select Flow > Delay
Set Time : 0.5

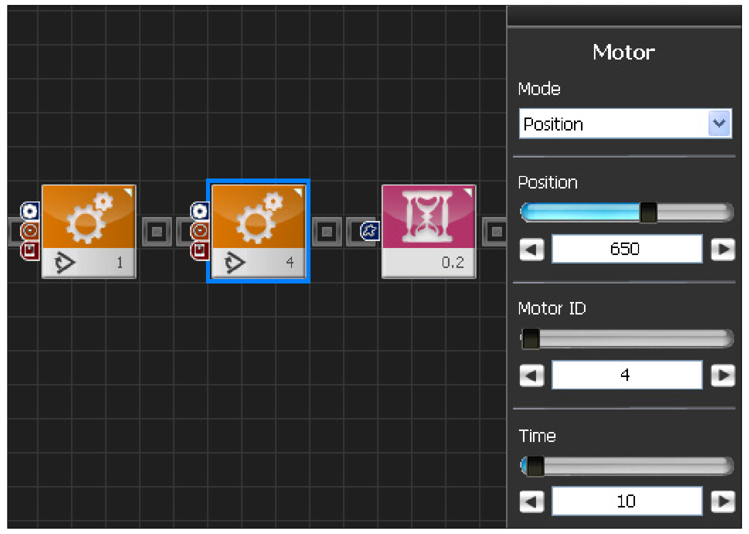


39 Set Motor ID1 (Upper Right Arm)

9th Stage
: Lower arm to 45 degrees angle.

Lower arm to 45 degrees angle to return to attention posture.

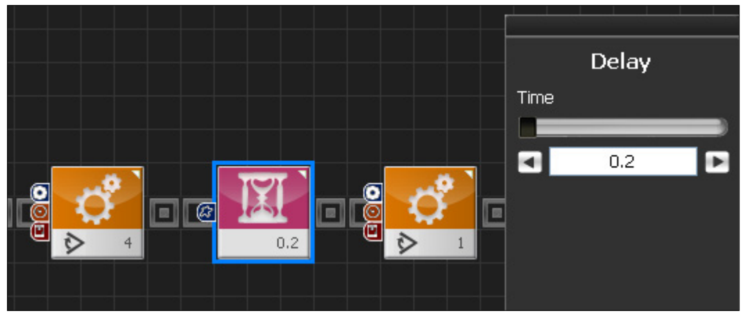
Select Motion > Motor
Set Mode : Position
Set Position : 374
Set Motor ID : 1
Set Time : 10, Motor will move to required position in 0.112s



40 Set Motor ID4 (Upper Left Arm)

Lower upper left arm to 45 degrees angle.

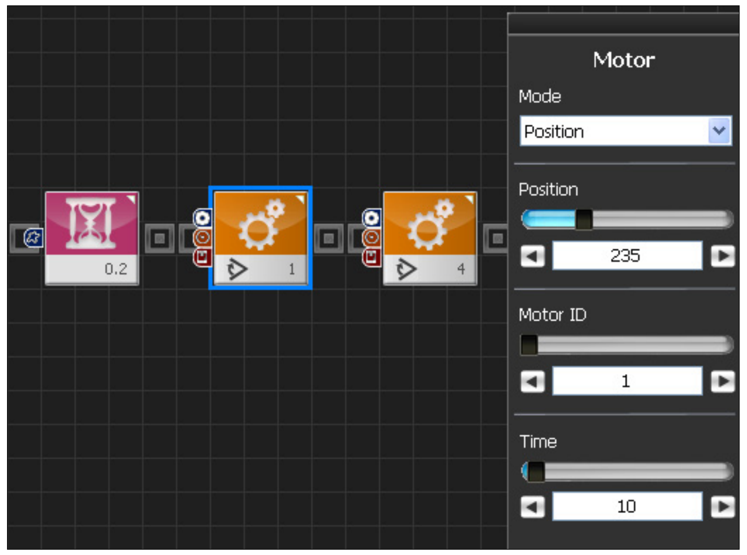
Select Motion > Motor
Set Mode : Position
Set Position : 650
Set Motor ID : 4
Set Time : 10, Motor will move to required position in 0.112s



41 Delay

Delay 0.2s.

Select Flow > Delay
Set Time : 0.2

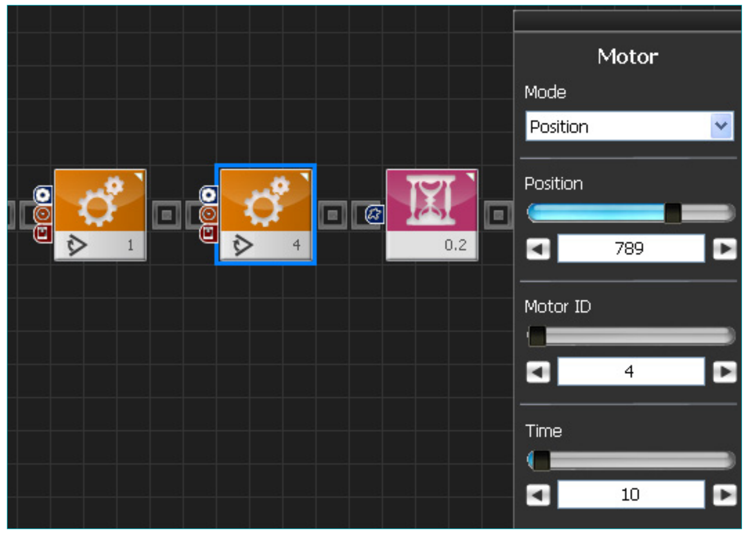


42 Set Motor ID1 (Upper Right Arm)

10th stage: Dance complete

Return to attention posture.

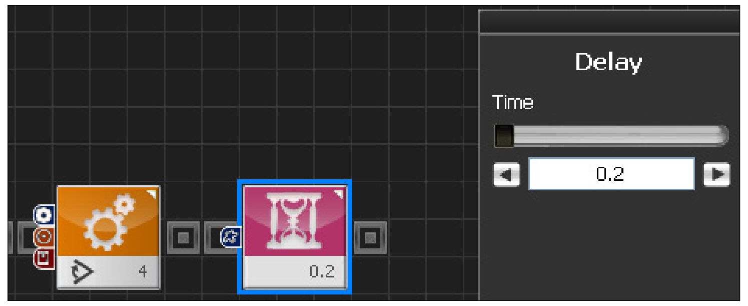
Select Motion > Motor
Set Mode : Position
Set Position : 235



43 Set Motor ID4 (Upper Left Arm)

Return to attention posture.

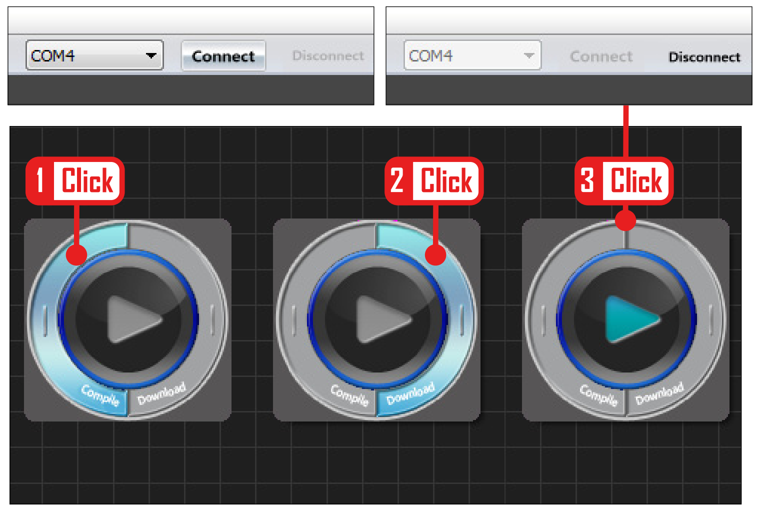
Select Motion > Motor
Set Mode : Position
Set Position : 789
Set Motor ID : 4
Set Time : 10, Motor will move to required position in 0.112s



44 Delay

Delay 0.2s until the motion ends.

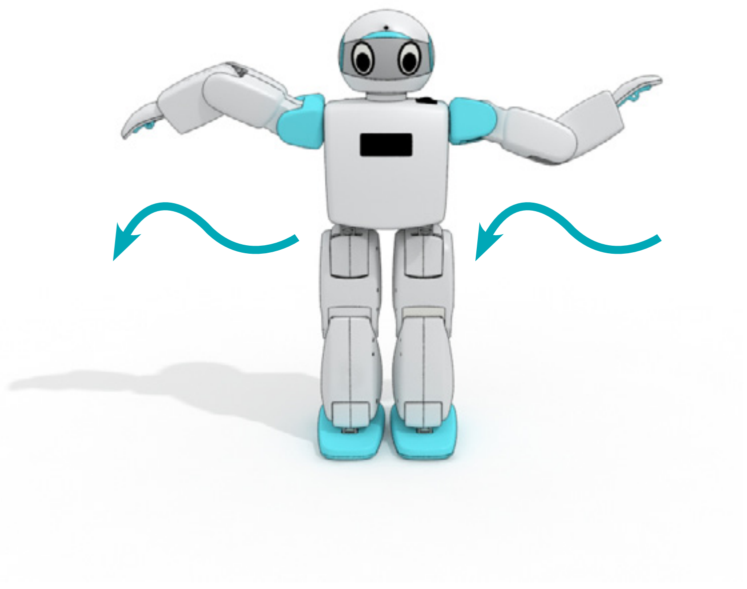
Select Flow > Delay
Set Time : 0.2



45 Download

After programming is completed.
Compile -> Download -> Run.

Select the appropriate COMPORT or USB-to-Serial number located on the PC and press Connect to open the serial port.
Click ‘Compile’ and then click ‘Download’ on the right to download the program to the robot if no compile error is found.
Click “Run” button (arrow in the middle) after completing the download.



46 Robot Motion

Wave dance will star from the lett arm.

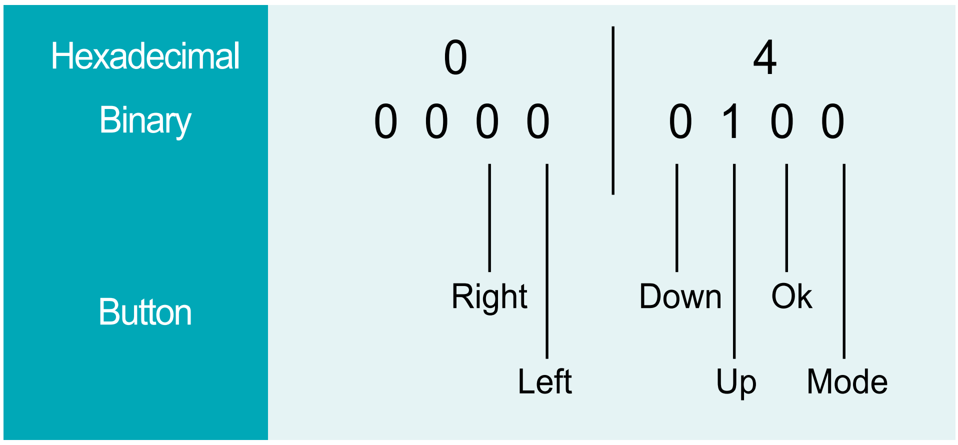
08-3 Programming Individual Module
Button, LED

Example Description

This example uses the buttons on the DRC controller to turn on/off the LED.

In order to program the button and the LED, understanding of registers corresponding.

16 = 2^4

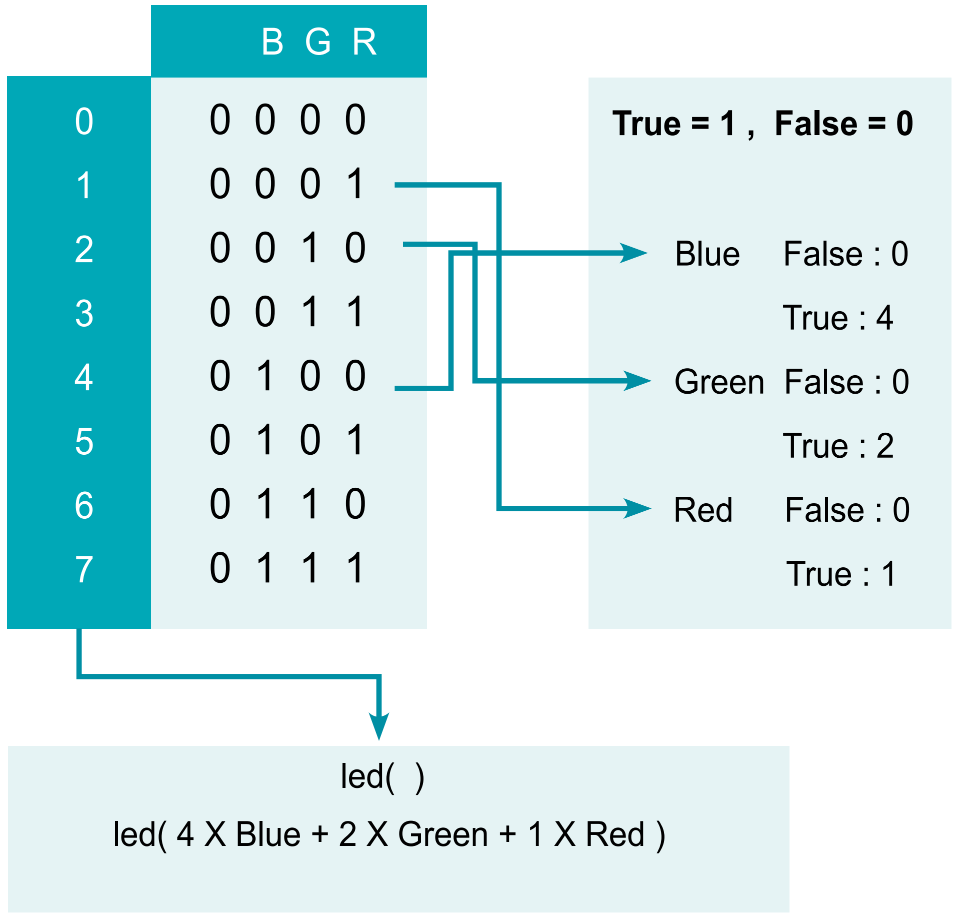


EX)	Right	2	0	h
		0 0 1 0	0 0 0 0	
	Down	0	8	h
		0 0 0 0	1 0 0 0	

Button

DRC has 6 buttons and pressed button is expressed by a 1 Byte. 1 Byte is made up of 8 Bits so 1Byte is able to carry 81s and 0s. 6 bits are required to express pressed (1) and released (0) status of 6 DRC buttons. As shown in the diagram above, each button is matched with a single bit. Pressed button is expressed in 1s and 0s and it is shown in hexadecimal format at bottom right side of the button module. Pressed 'right'; button has value of 00100000 or 20h when converted to hexadecimal format (h refers to hexadecimal). Pressed 'down' button has value or 00001000 or 08h in hexadecimal format.

For something more complicated, pressed 'up+down' button has value of 00001100 or 0Ch in hexadecimal format.



		B	G	R
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1

True = 1 , False = 0

Blue	False : 0
	True : 4
Green	False : 0
	True : 2
Red	False : 0
	True : 1

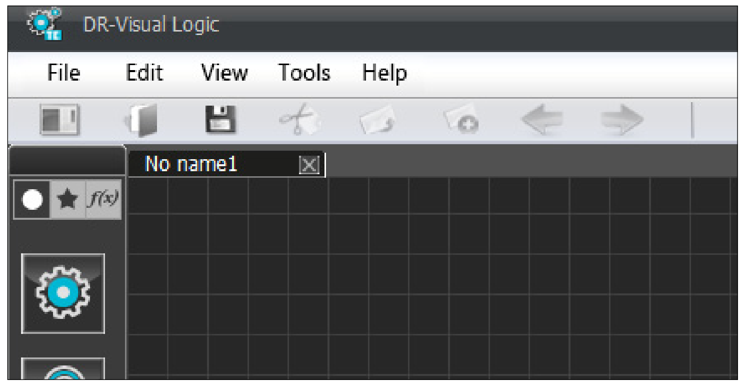
led( )

led(4 X Blue + 2 X Green + 1 X Red)

LED

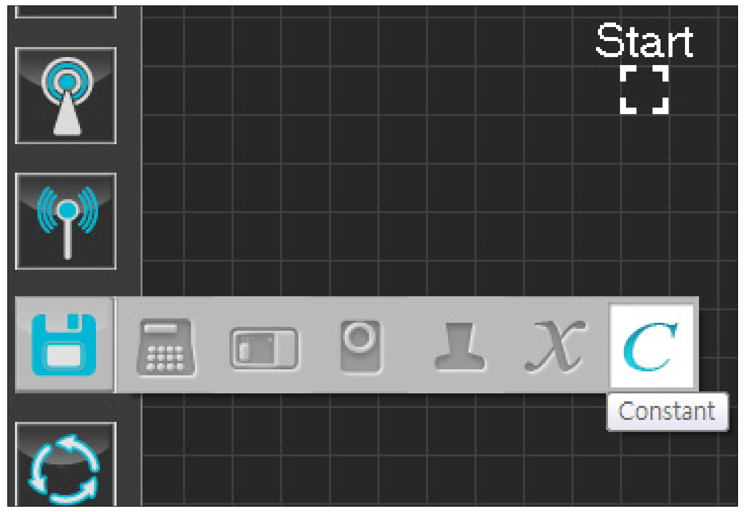
DRC has seven LEDs but only three can be controlled by the task mode. Three bits are required to express on/off status of the three LEDs; Red, Green, Blue. As shown in the diagram above, each LED (Red, Green Blue) is matched with a bit in an ascending order from the lowest bit of the byte to the highest. LED lights up when the LED value is used as an input of the LED module. All LEDs are turned off when the input value is 0(00000000) and they are turned on when the input value is 7(00000111). Blue in binary format is 4, Green 2, and Red 1. When on/off state of each individual LED is determined by the value (true, false) of the variables Blue, Green, and Red, it is possible to control the LEDs by their variable names using 4 x Blue + 2 x Green + 1 x Red as the input of the LED module. For example, when Blue and Green is 'true' and Red 'false, it becomes 4 x Blue + 2 x Green + 1 x Red = 6. 6 in binary format is 00000110. Green and Blue LED will light up when this value is used as an input of the LED module.

Use the basic principals from above to program the Buttons and LEDs.



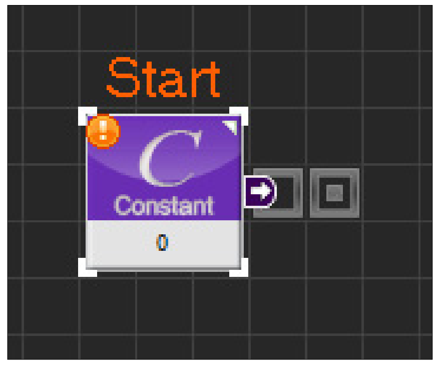
01 Create New File

Click on the left most icon in the tools to create new dts file. Select HOVIS Lite/ Eco as the Robot.



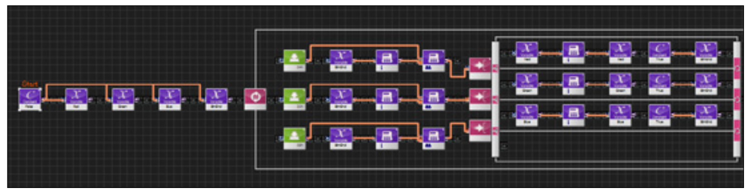
02 Select Module

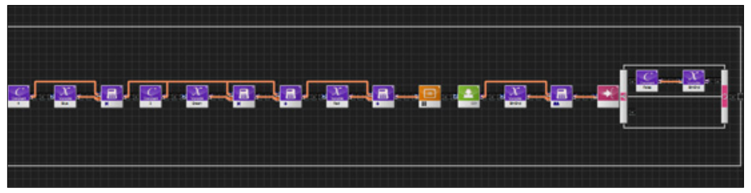
Prior to placing the module, click to select the module from the module bar located on the left side.
Click Data > Constant module



03 Place Module

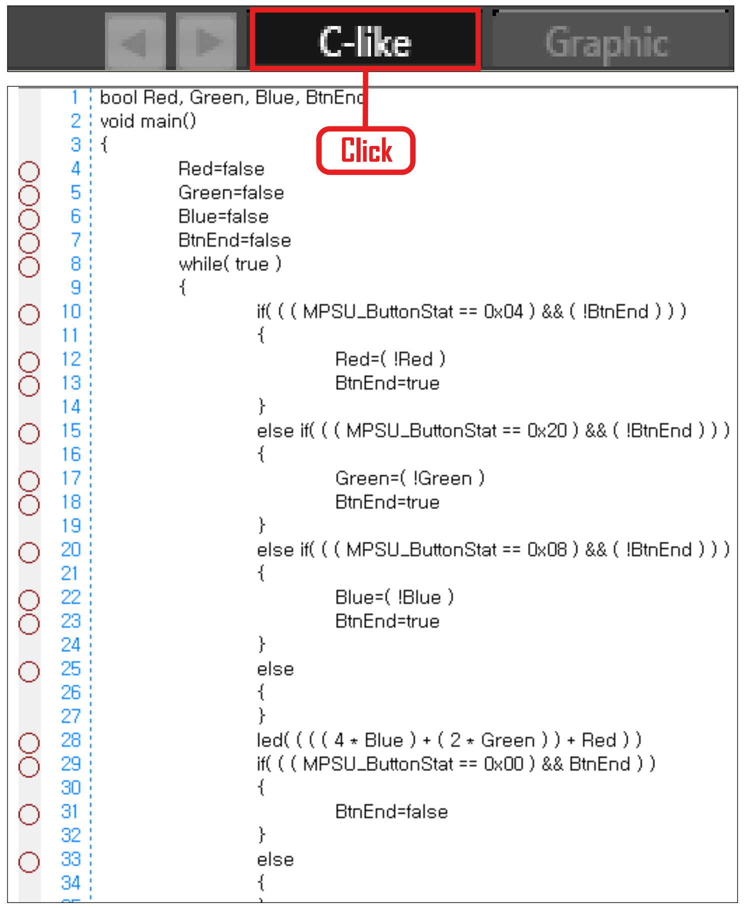
Drag and dock the module to the Start Point to activate the module.
Active module will turn to color





04 Entire Program

Entire program using Button and LED.

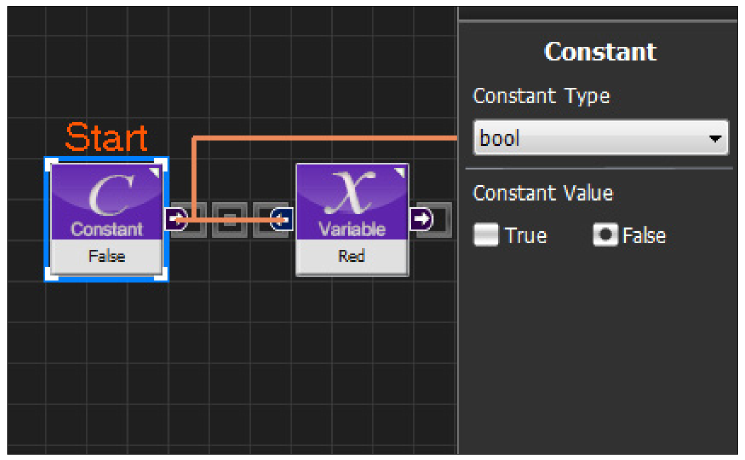


```
bool Red, Green, Blue, BtnEnd
void main()
{
        Red=false
        Green=false
        Blue=false
        BtnEnd=false
        while( true )
        {
                if( ( ( MPSU_ButtonStat == 0x04 ) && ( !BtnEnd ) ) )
                {
                        Red=( !Red )
                        BtnEnd=true
                }
                else if( ( ( MPSU_ButtonStat == 0x20 ) && ( !BtnEnd ) ) )
                {
                        Green=( !Green )
                        BtnEnd=true
                }
                else if( ( ( MPSU_ButtonStat == 0x08 ) && ( !BtnEnd ) ) )
                {
                        Blue=( !Blue )
                        BtnEnd=true
                }
                else
                {
                }
                led( ( ( ( 4 * Blue ) + ( 2 * Green ) ) + Red ) )
                if( ( ( MPSU_ButtonStat == 0x00 ) && BtnEnd ) )
                {
                        BtnEnd=false
                }
                else
                {
```

05 Viewing C-Like

Click "C-Like" tab near the top right to open the program source code window as shown in the left.

Since codes are very similar to the C language structure, studying the codes will help the user become familiar with the C language as well. Clicking on the module results in cursor jumping to the matching source code line in the C-Like window. This feature enables the user to easily see the codes generated by the module.



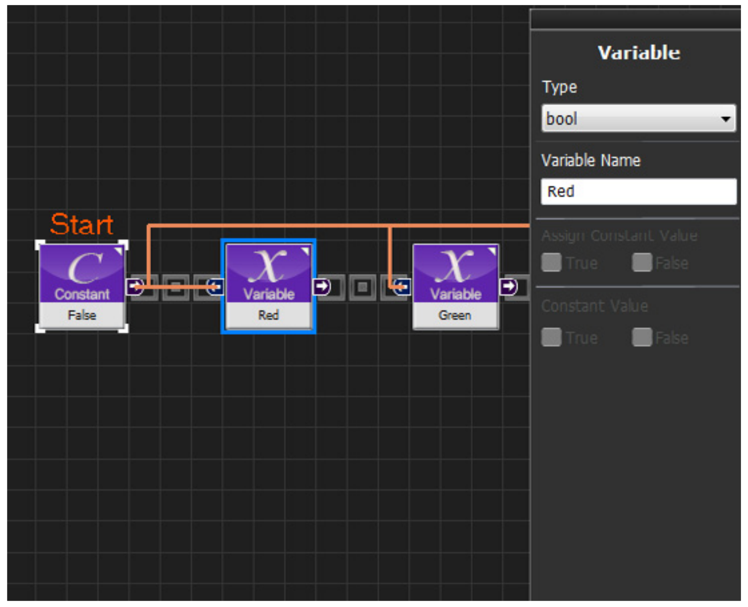
Constant

Constant Type: bool

Constant Value: True [✓ False]

06 Set Constant

Set constant to initialize all LED variables to off (false).
Change Constant Type in Constant module properties to bool.
Set Constant Value to False
This value is sent to the input pin of the next module through the output pin.



Variable

Type: bool

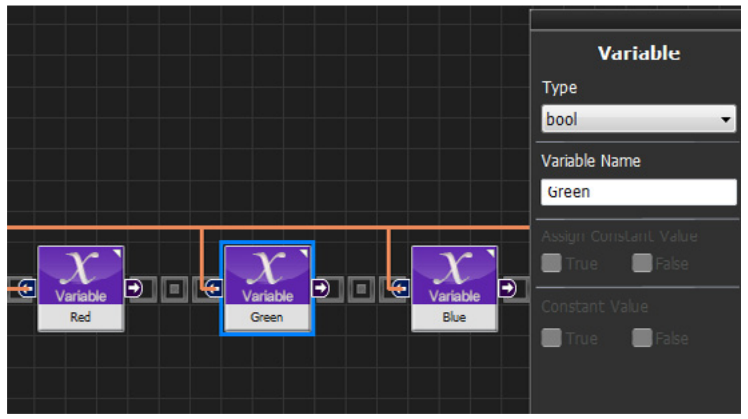
Variable Name: Red

Assign Constant Value: True False

Constant Value: True False

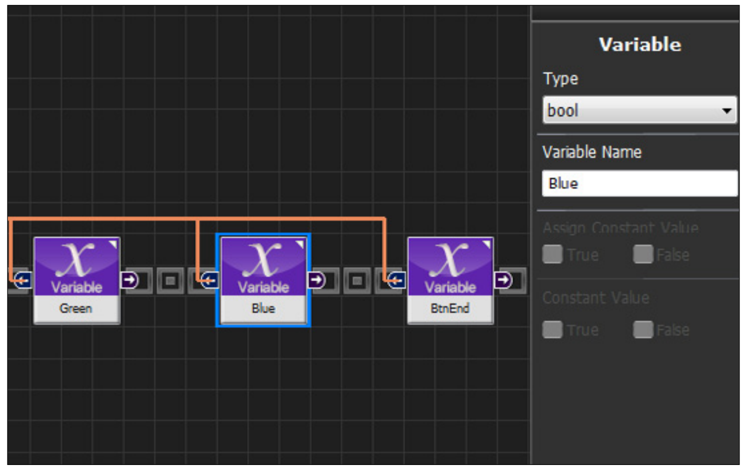
07 Initialize Red Variable

Create Variable module to initialize the variable that will hold the Red LED value.
Since variable will hold two values TRUE and FALSE, bool type variable will be used.
Select Data > Variable and place it after the previous module.
Set Type : bool
Enter Variable Name : Red
Use the connector to connect the output pin of the Constant module to the input pin of the Variable module. Value 'False' will be entered in the variable Red.


Variable
Type: bool
Variable Name: Green
Assign Constant Value: True False
Constant Value: True False

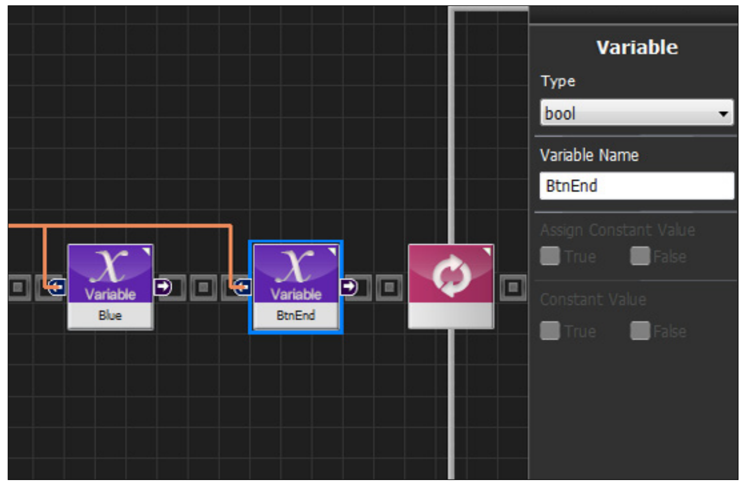
08 Initialize Green Variable

Create Variable module to initialize the variable that will hold the Green LED value. Since variable will hold two values TRUE and FALSE, bool type variable will be used.
Select Data > Variable and place it after the previous module.
Set Type : bool
Enter Variable Name : Green
Use the connector to connect the output pin of the Constant module to the input pin of the Variable module. Value 'False' will be entered in the variable Green.


Variable
Type: bool
Variable Name: Blue
Assign Constant Value: True False
Constant Value: True False

09 Initialize Blue Variable

Create Variable module to initialize the variable that will hold the Blue LED value. Since variable will hold two values TRUE and FALSE, bool type variable will be used.
Select Data > Variable and place the module after the previous module.
Set Type : bool
Enter Variable Name : Blue
Use the connector to connect the output pin of the Constant module to the input pin of the Variable module. Value 'False' will be entered in the variable Blue.

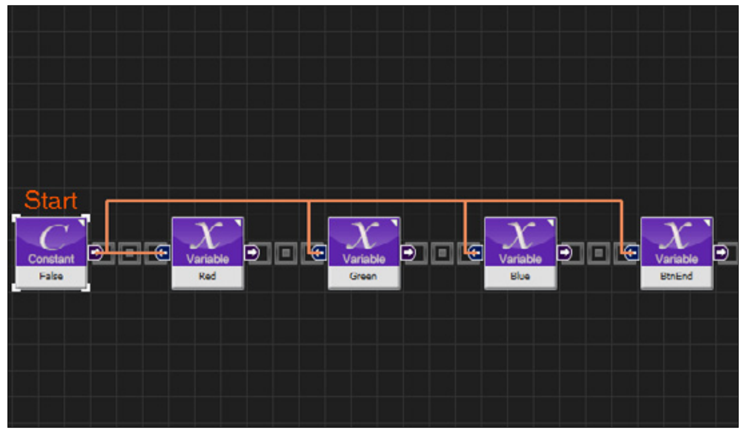

Variable
Type: bool
Variable Name: BtnEnd
Assign Constant Value: True False
Constant Value: True False

10 Intialize BtnEnd Variable

BtnEnd variable saves the end of the required motion when button is pressed. This program changes the LED variable value when button is pressed within the loop. However, due to the speed of the loop, variable value changes several times during single button press. BtnEnd variable is declared to save the end of variable value change so that single button press will change the variable value only once.

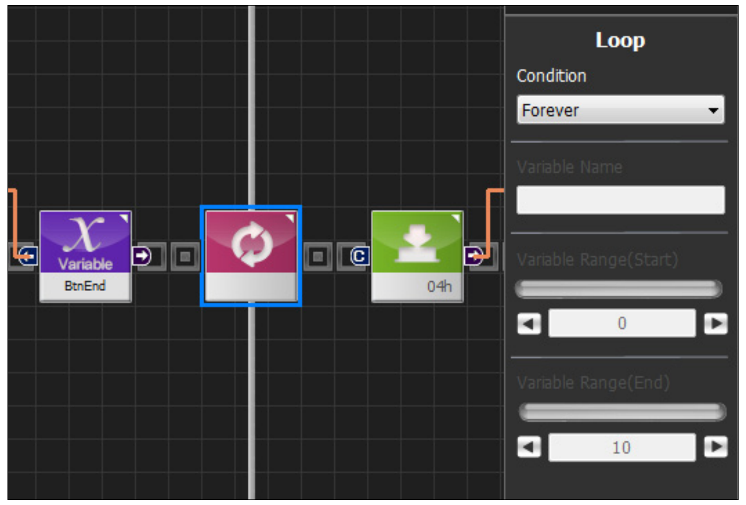
Select Data > Variable and place the module after the previous module.

Enter Variable Name : BtnENd
Use the connector to connect the output pin of the Constant to the input pin of the Variable module.



11 Variable Initialization Complete

Red, Green, Blue, BtnEnd variables initialized to False. As shown, output pin from a single module can be connected to input pins from multiple modules.

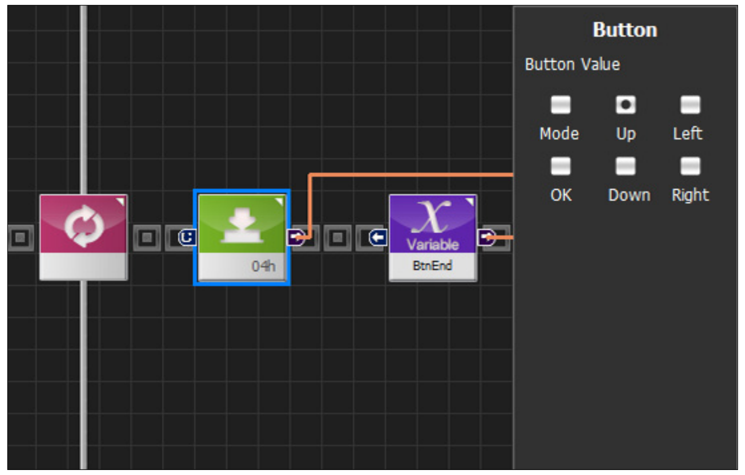

Loop
Condition: Forever
Variable Name
Variable Range(Start): 0
Variable Range(End): 10

12 Loop

Create main loop module.

Select Flow > Loop and place the module after the previous module.

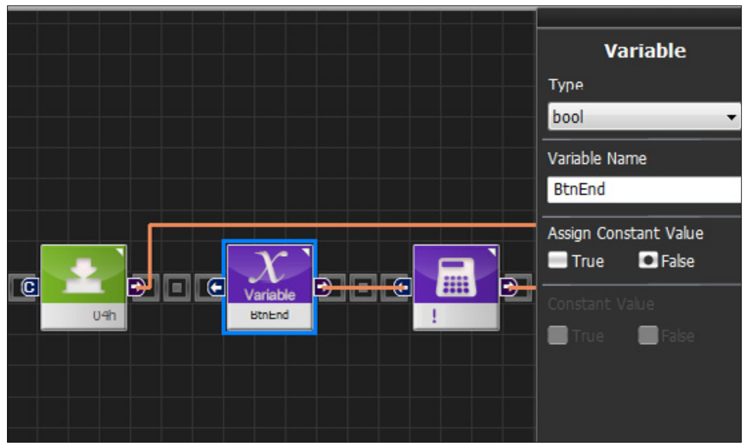
Select Condition : Forever, infinite loop without any condition.


Button
Button Value: Mode Up Left OK Down Right

13 Up Button

Create button module. Output value of this module becomes True when selected button is pressed and False otherwise.
When Up Button is selected, True when Up Button is pressed and False otherwise.

Select Communication > Button module and place it within the Loop.
Set Button Value to Up.
Hexadecimal number 04h is shown on the module.


Variable

Type: bool

Variable Name: BtnEnd

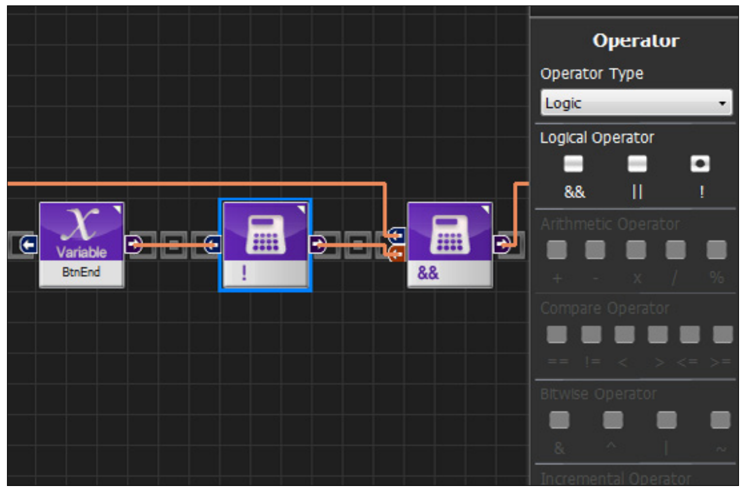
Assign Constant Value: True ■ False

Constant Value: True False

14 BtnEnd

Read BtnEnd variable value for use in the conditional statement.

Select Data > Variable and place the module after the previous module
Set Type : bool
Enter Variable Name : BtnEnd.
BrnEnd from previous section can be copied and pasted.


Operator

Operator Type: Logic

Logical Operator: && || ■ !

Arithmetic Operator

Compare Operator

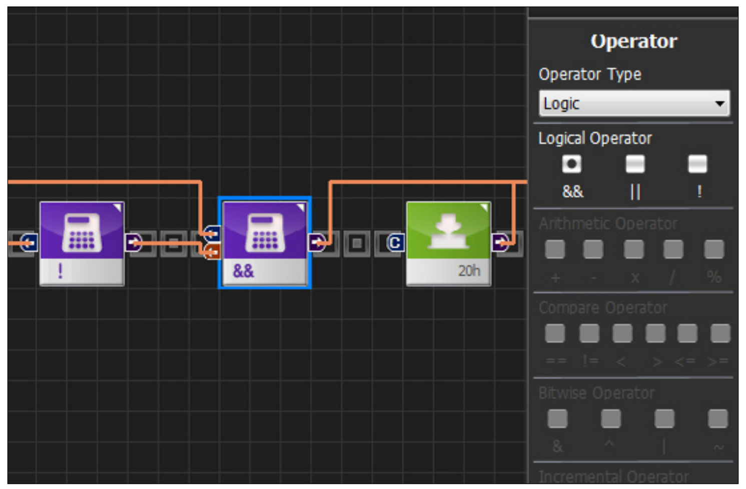
Bitwise Operator

Incremental Operator

15 NOT Operation

Apply Not operator to reverse the value of BtnEnd.

Select Data > Operator module
Set Operator Type : Logic
Select Logical Operator : (Logical NOT)
Connect BtnEnd output pin to NOT module input pin


Operator

Operator Type: Logic

Logical Operator: ■ && || !

Arithmetic Operator

Compare Operator

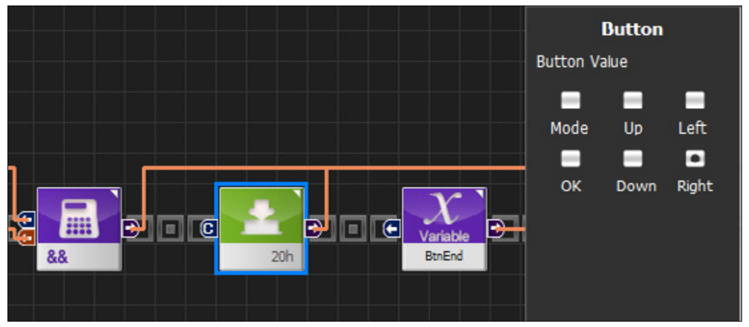
Bitwise Operator

Incremental Operator

16 AND Operation

Apply AND operator to set output value to TRUE when Up button is pressed and BtnEnd value is false (True with NOT applied).

Select Data > Operator module.
Select Operator Type : Logic.
Select Logical Operator : &&(Logical AND).
Connect output pin of Button module from step 13 and NOT module from step 15 to AND module input pin.

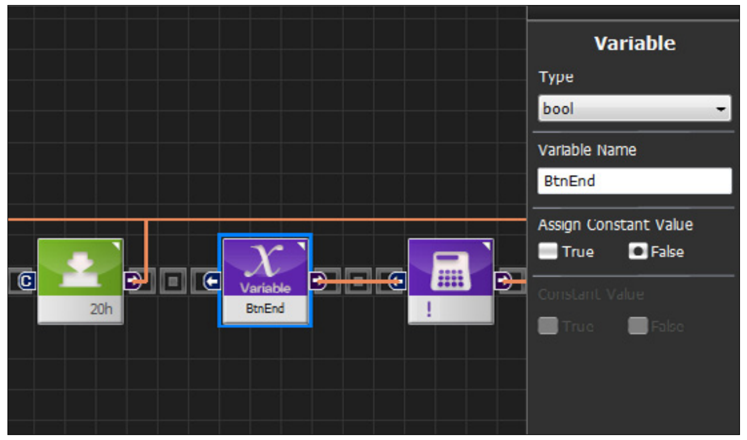

Button

Button Value: Mode Up Left OK Down ■ Right

17 Right Button

Create button module and select Right button

Select Communication > Button and place the module after the previous module.
Set Button Value as Right
Hexadecimal number 20h is shown on the module.


Variable

Type: bool

Variable Name: BtnEnd

Assign Constant Value: True ■ False

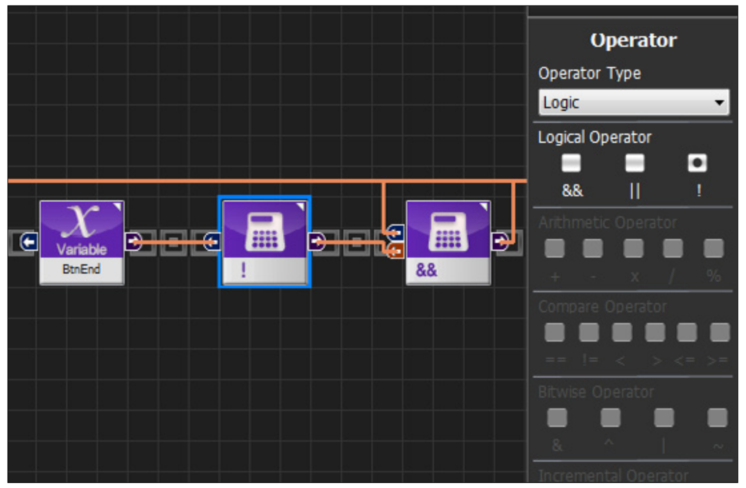
Constant Value: True False

18 BtnEnd

Read BtnEnd variable value for use in the conditional statement.

Select Data > Variable and place the module after the previous module.

Seletct Type : bool
Enter Variable Name : BtnEnd
BtenEnd from previous step can be copied and pasted.


Operator

Operator Type: Logic

Logical Operator: && || ■ !

Arithmetic Operator

Compare Operator

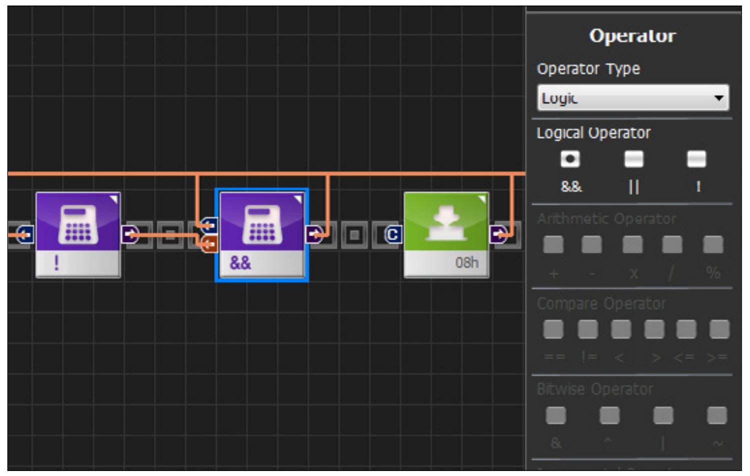
Bitwise Operator

Incremental Operator

19 NOT Operation

Apply Not operator to reverse the value of BtnEnd.

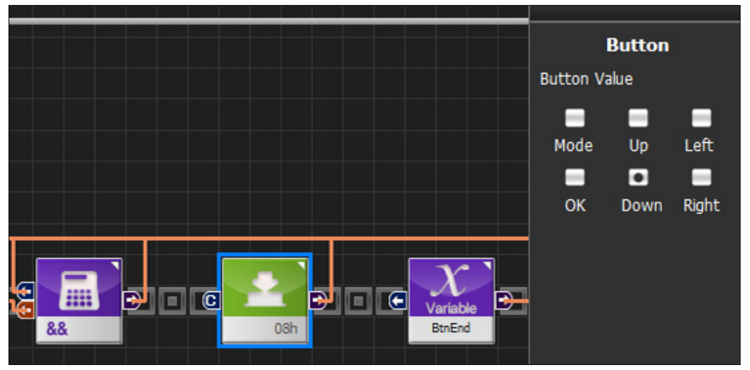
Select Data > Operator module
Set Operator Type : Logic
Select Logical Operator : (Logical NOT)
Connect BtnEnd output pin to NOT module input pin.



20 AND Operation

Apply AND operator to set output value to TRUE when Right button is pressed and BtnEnd value is false (True with NOT applied).

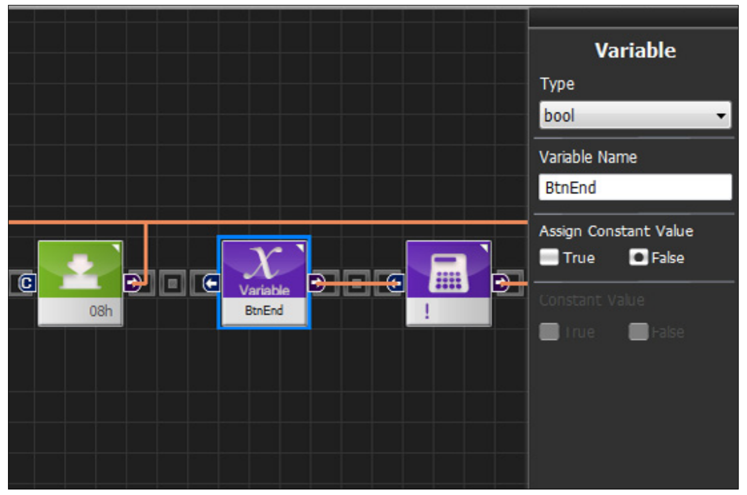
Select Data > Operator module.
Select Operator Type : Logic.
Select Logical Operator : &&(Logical AND).
Connect output pin of Button module from step 17 and NOT module from step 19 to AND module input pin.



21 Down Button

Create button module and select Down button

Select Communication > Button and place the module after the previous module.
Set Button Value as Down
Hexadecimal number 08h is shown on the module.

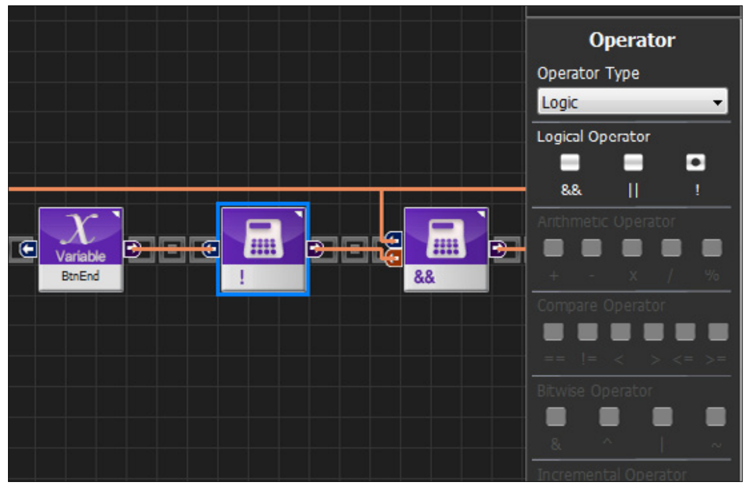


22 BtnEnd

Read BtnEnd variable value for use in the conditional statement.

Select Data > Variable and place the module after the previous module.

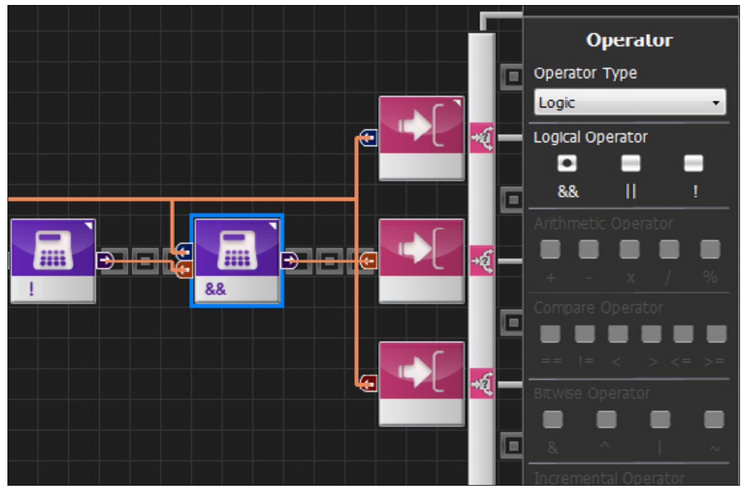
Seletct Type : bool
Enter Variable Name : BtnEnd
BtenEnd from previous step can be copied and pasted.



23 NOT Operation

Apply Not operator to reverse the value of BtnEnd.

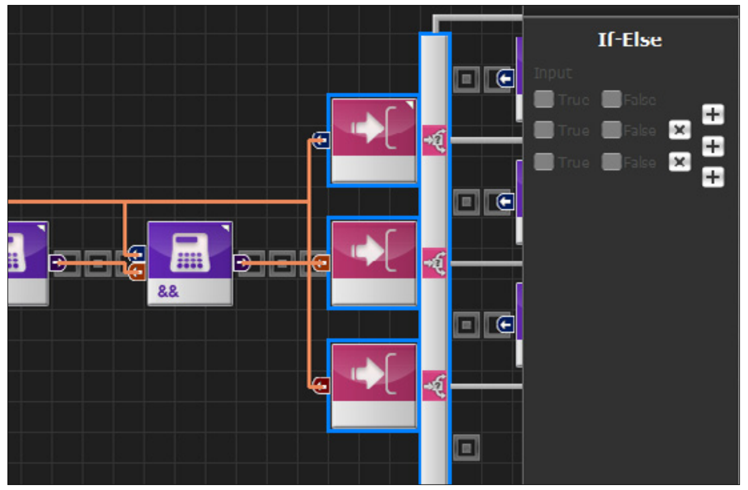
Select Data > Operator module
Set Operator Type : Logic
Select Logical Operator : (Logical NOT).
Connect BtnEnd output pin in step 22 to NOT module input pin.



24 AND Operation

Apply AND operator to set output value to TRUE when Down button is pressed and BtnEnd value is false (True with NOT applied).

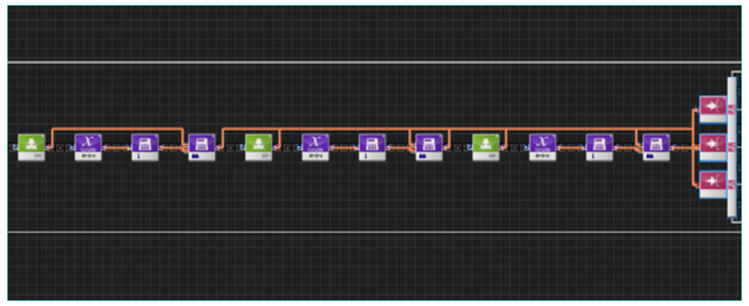
Select Data > Operator module.
Select Operator Type : Logic.
Select Logical Operator
: &&(Logical AND).
Connect output pin of Button module from step 21 and NOT module from step 23 to AND module input pin.



25 Create If-Else Statement

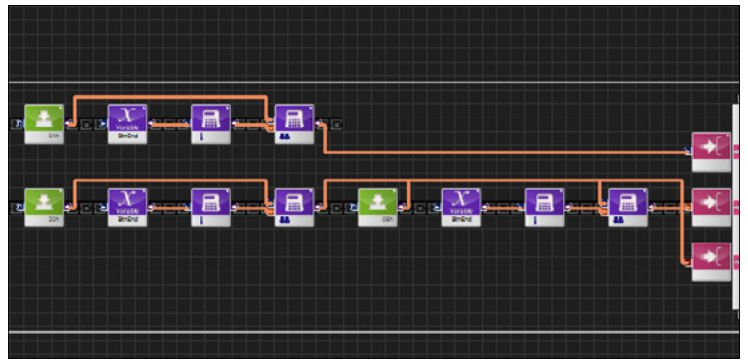
Create If-Else statement and connect previously created conditional statement to the If-Else statement.

Select Flow > If-Else module
Click + shaped icon twice to add two more conditional statements.
Connect the AND operator output pin from step 12 to the first input pin.
Connect output pin from step 20 to the second input pin.
Connect output pin from step 24 to the third input pin.



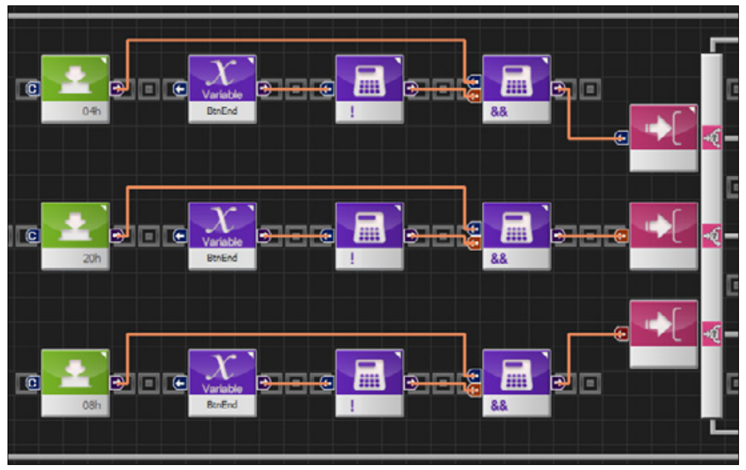
26 Conditional Statement Completed

If-Else conditional statement using Up button, Right button, and Down button has been completed. Modules should be placed so that it is to recognize the connections.



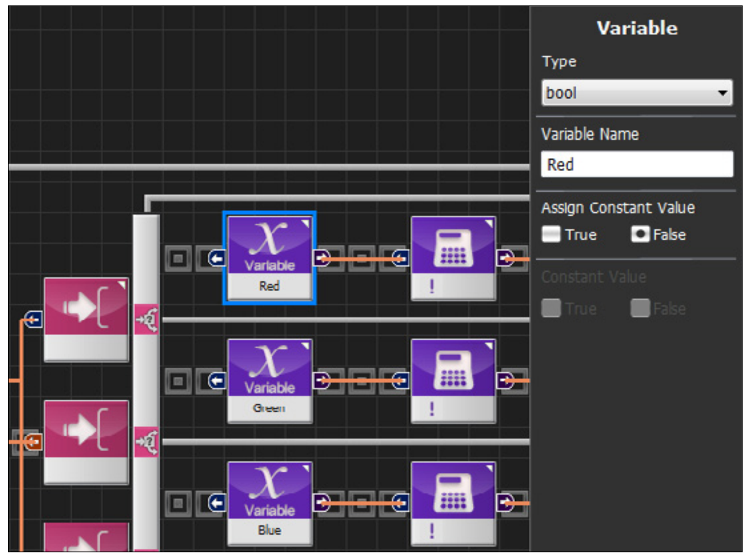
27 Module Placement

Place four modules created in steps 13 ~16 above the program line. With the output pins connected, modules will remain active even though they are placed outside the line.



28 Module Placement

Place four modules created in steps 21 ~24 below the program line. Placing the modules in such a way makes it easy to distinguish which conditional statement is connected to which input pin.

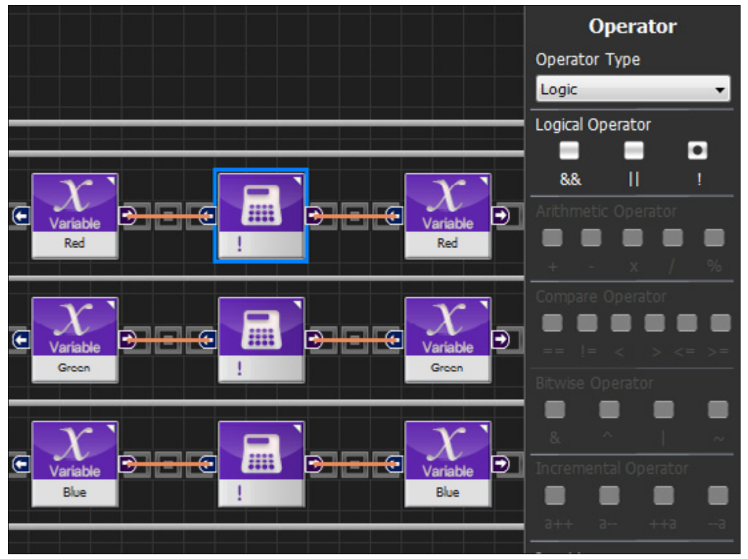


Variable
Type
bool
Variable Name
Red
Assign Constant Value
True False
Constant Value
True False

29 Read Red Variable

Add Modules to be executed within the If-Else module. Content of the highest program line will be executed if the first input pin connected to Up button is True.
In order to change the value of the variable Red, first create and read the value from the Red variable module.

Select Data > Variable and place it in the highest program line in the If-Else statement.
Set Type : bool
Enter Variable Name : Red
Red variable from previous section can be copied and pasted as well.

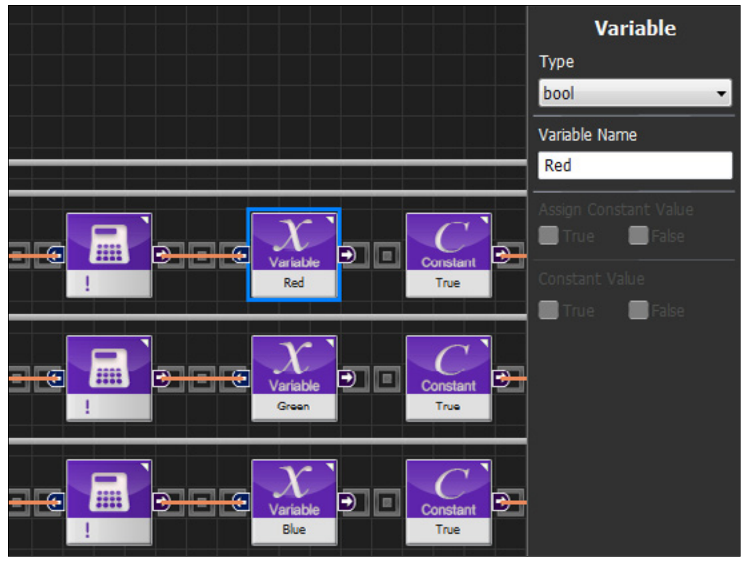


Operator
Operator Type
Logic
Logical Operator
&& || !
Arithmetic Operator
+ - x / %
Compare Operator
== != < > <= >=
Bitwise Operator
& ^ | ~
Incremental Operator
a++ a-- ++a --a

30 NOT Operation

Apply NOT operator to reverse the value of RED.

Select Data > Operator
Set Operator Type : Logic
Select Logical Operator
: !(Logical NOT)
Connect Red output pin from step 29 to the NOT module input pin.



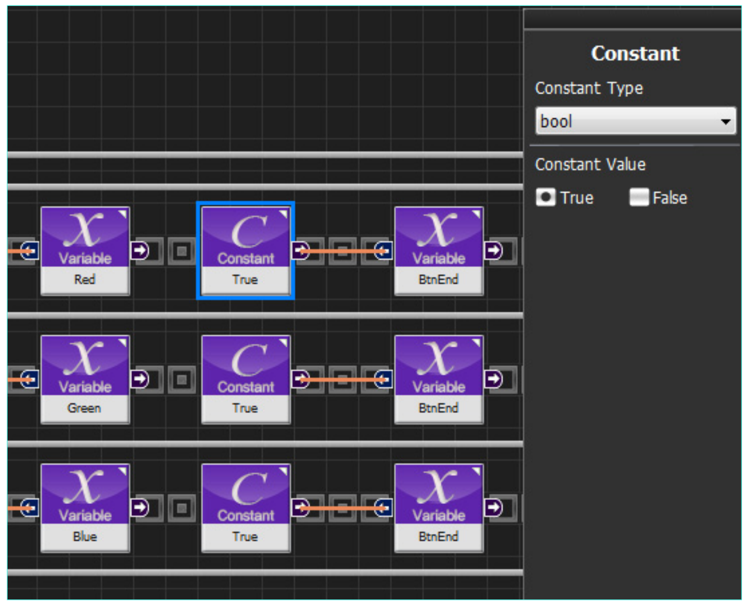
Variable
Type
bool
Variable Name
Red
Assign Constant Value
True False
Constant Value
True False

31 Substitute RED Variable Value

Substitute RED value with reversed value.

Select Data > Variable and place the module after the previous module.

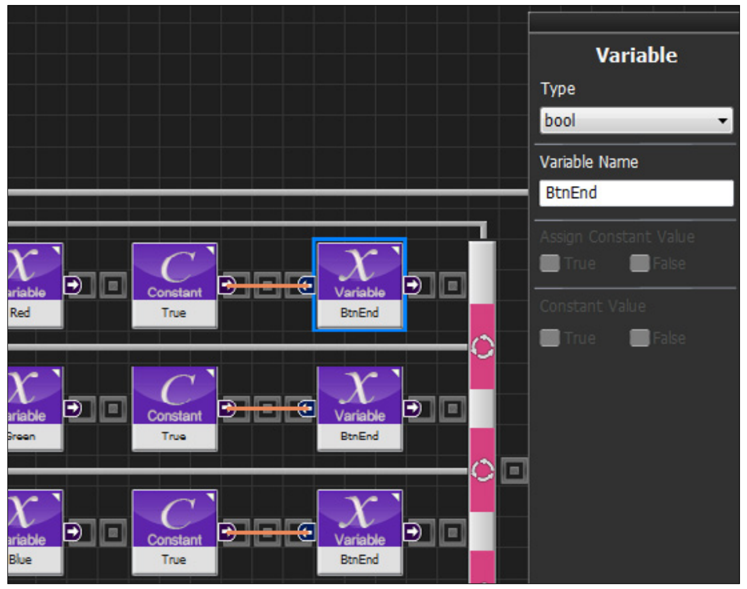
Set Type : bool.
Enter Variable Name : Red
Connect the NOT module output pin from step 30 to the Red variable input pin and substitute the Red value with reversed value.



32 Generate 'True'

Create Constant module and generate 'True' to be substituted into the BtnEnd variable.

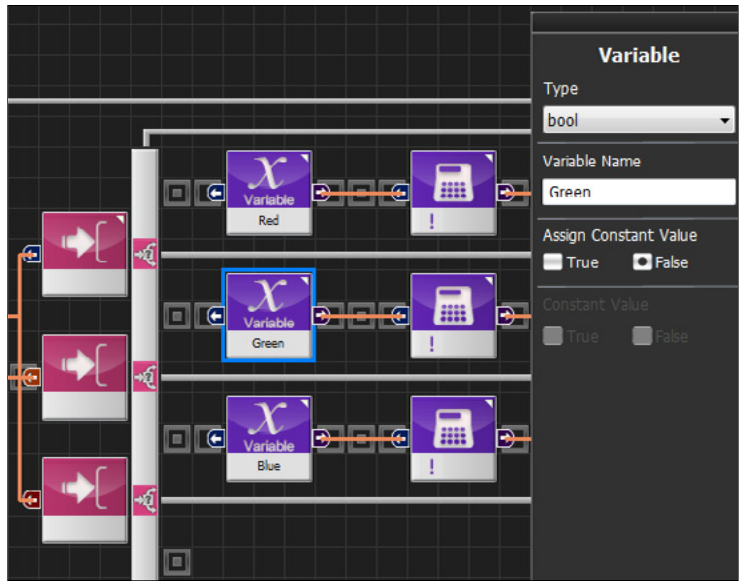
Select Data > Constant and place the module after the previous module. Change one of the Constant properties Constant Type to bool.
Set Constant Value to True.



33 Substitute "True" In BtnEnd

Substitute "True" into BtnEnd to express end of processing Up button press. When BtnEnd value is True, conditional statements formed in steps 13 ~ 16 cannot be True and therefore Red value will not change until button is released.

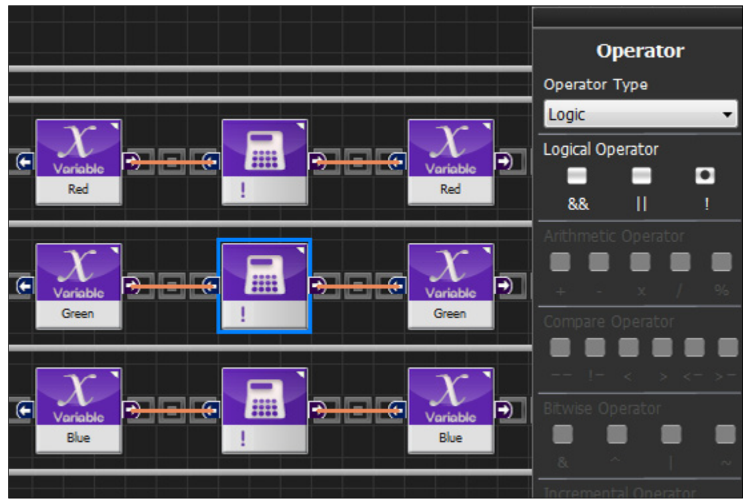
Selelct Data > Variable and place the module after the previous module.
Set Type : bool
Enter Variable Name : BtnEnd
Connect Constant output pin in step 32 to the BtnEnd input pin.



34 Read Green Variable

Content of the second program line will be processed if first input pin is False and second input pin is True. In order to change the value of the variable Green, first create and read the value from the Green variable module.

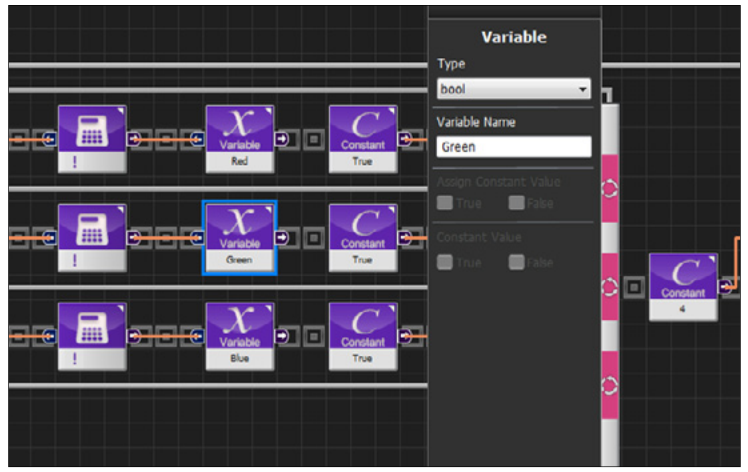
Select Data > Variable and place the module in the second program line of the If-Else statement.
Set Type : bool
Enter Variable name : Green.



35 NOT Operation

Apply NOT operator to reverse the Green value.

Select Data > Operator.
Set Operator Type : Logic.
Select Logical Operator : !(Logical NOT).
Connect Green output pin from step 34 to NOT module input pin.

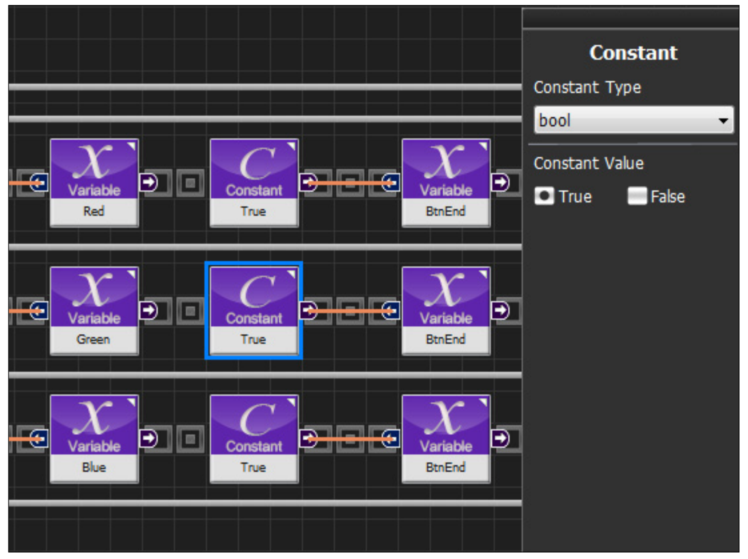


36 Substitute Green Variable Value

Substitute Green value with reversed value.

Select Data > Variable and place the module after the previous module.

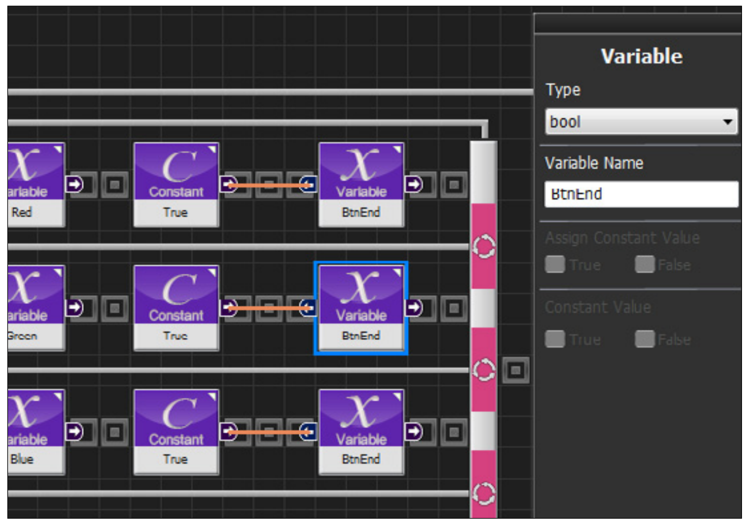
Set Type : bool.
Enter Variable Name : Green
Connect the NOT module output pin from step 35 to the Green variable input pin and substitute the Green value with reversed value.



37 Generate 'True'

Create Constant module and generate 'True' to be substituted into the BtnEnd variable.

Select Data > Constant and place the module after the previous module.
Change one of the Constant properties Constant Type to bool.
Set Constant Value to True.

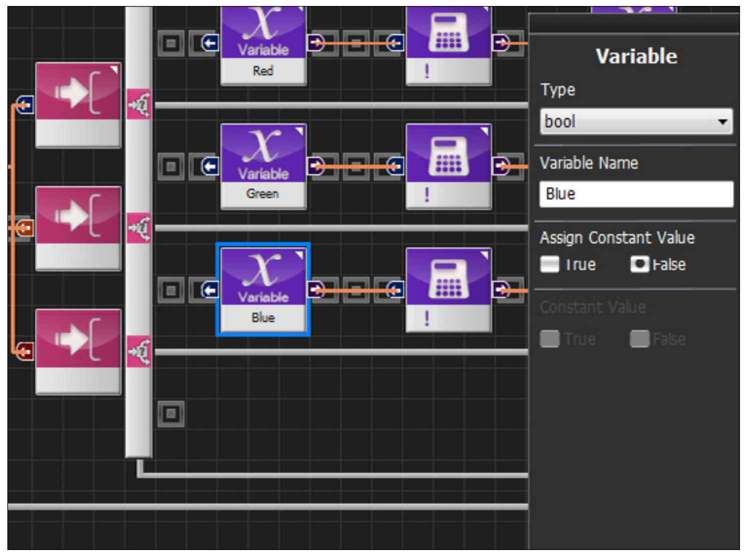


Variable
Type: bool
Variable Name: BtnEnd
Assign Constant Value: True False
Constant Value: True False

38 Substitute “True” In BtnEnd

Substitute “True” into BtnEnd to express end of processing Right button press. When BtnEnd value is True, conditional statements formed in steps 17 ~ 20 cannot be True and therefore Green value will not change until button is released.

Selelct Data > Variable and place the module after the previous module.
Set Type : bool
Enter Variable Name : BtnEnd
Connect Constant output pin in step 37 to the BtnEnd input pin.

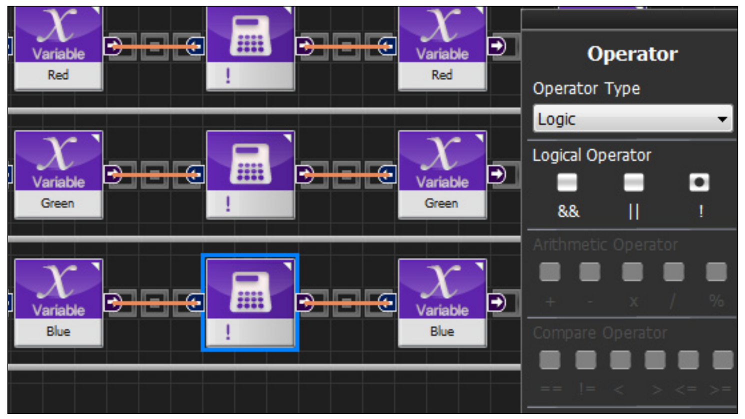


Variable
Type: bool
Variable Name: Blue
Assign Constant Value: True False
Constant Value: True False

39 Read Blue Variable

Content of the third program line will be processed if first and second input pin is False and third input pin is True. In order to change the value of the variable Blue, first create and read the value from the Blue variable module.

Select Data > Variable and place the module in the third program line of the If-Else statement.
Set Type : bool
Enter Variable name : Blue.

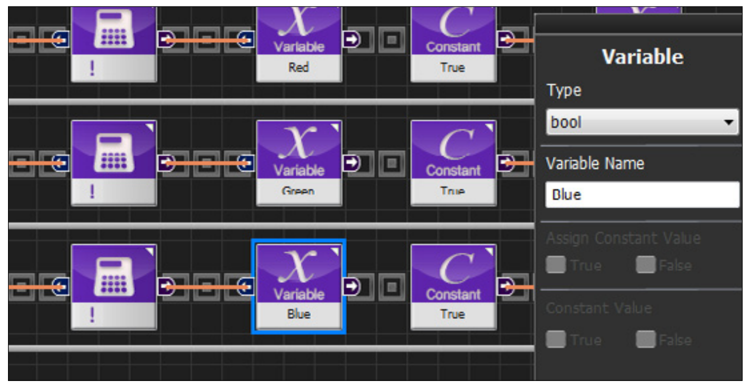


Operator
Operator Type: Logic
Logical Operator: && || !
Arithmetic Operator: + - x / %
Compare Operator: == != < > <= >=

40 NOT Operation

Apply NOT operator to reverse the Blue value.

Select Data > Operator.
Set Operator Type : Logic.
Select Logical Operator : !(Logical NOT).
Connect Blue output pin from step 39 to NOT module input pin.



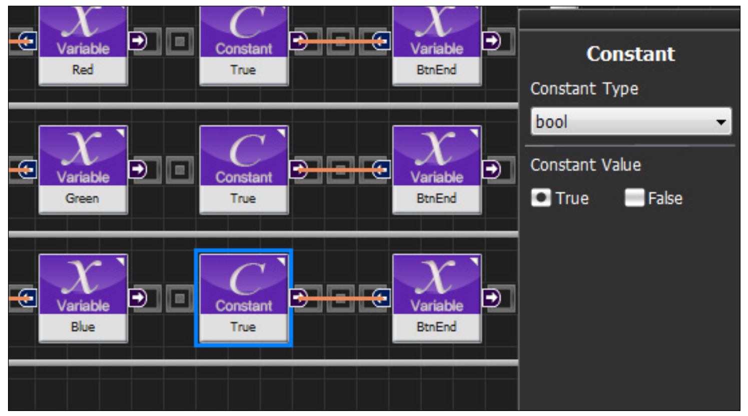
Variable
Type: bool
Variable Name: Blue
Assign Constant Value: True False
Constant Value: True False

41 Substitute Blue Variable Value

Substitute Blue value with reversed value.

Select Data > Variable and place the module after the previous module.

Set Type : bool.
Enter Variable Name : Blue
Connect the NOT module output pin from step 35 to the Blue variable input pin and substitute the Blue value with reversed value.

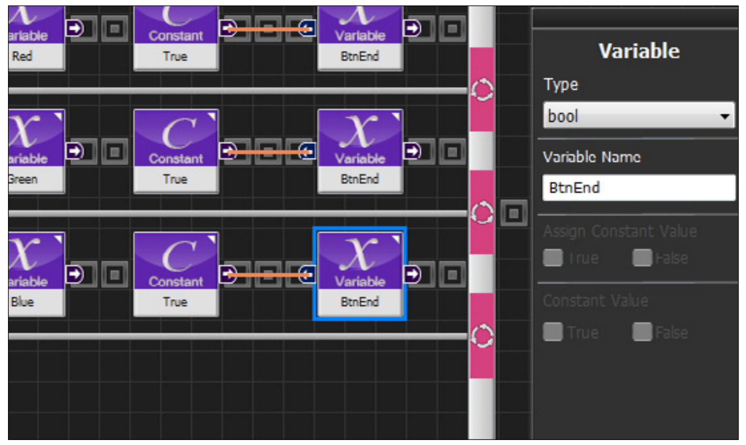


Constant
Constant Type: bool
Constant Value: True False

42 Generate ‘True’

Create Constant module and generate ‘True’ to be substituted into the BtnEnd variable.

Select Data > Constant and place the module after the previous module.
Change one of the Constant properties Constant Type to bool.
Set Constant Value to True.

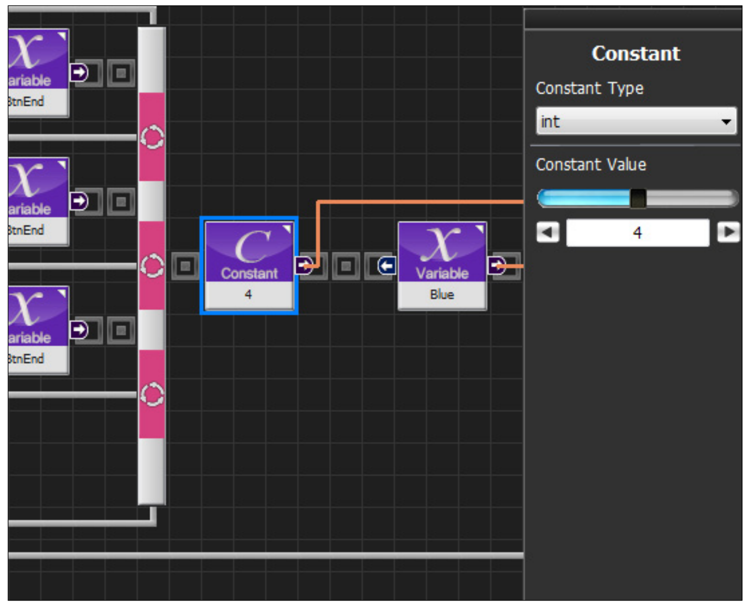


Variable
Type: bool
Variable Name: BtnEnd
Assign Constant Value: True False
Constant Value: True False

43 Substitute “True” In BtnEnd

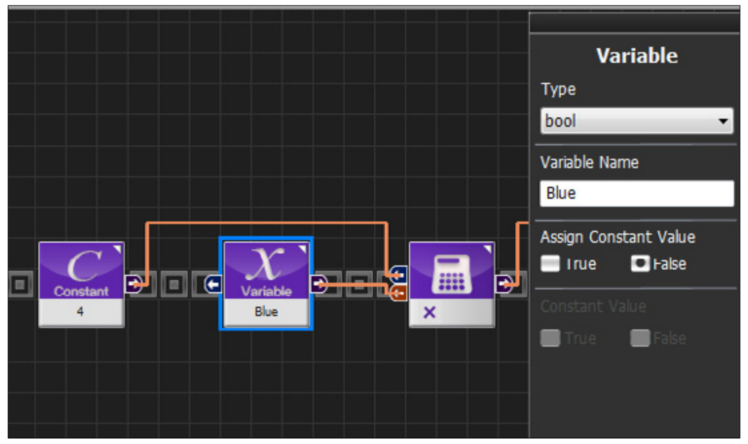
Substitute “True” into BtnEnd to express end of processing Down button press. When BtnEnd value is True, conditional statements formed in steps 21 ~ 24 cannot be True and therefore Blue value will not change until button is released.

Selelct Data > Variable and place the module after the previous module.
Set Type : bool
Enter Variable Name : BtnEnd
Connect Constant output pin in step 42 to the BtnEnd input pin


Constant
Constant Type
int
Constant Value
4

44 Add Constant 4

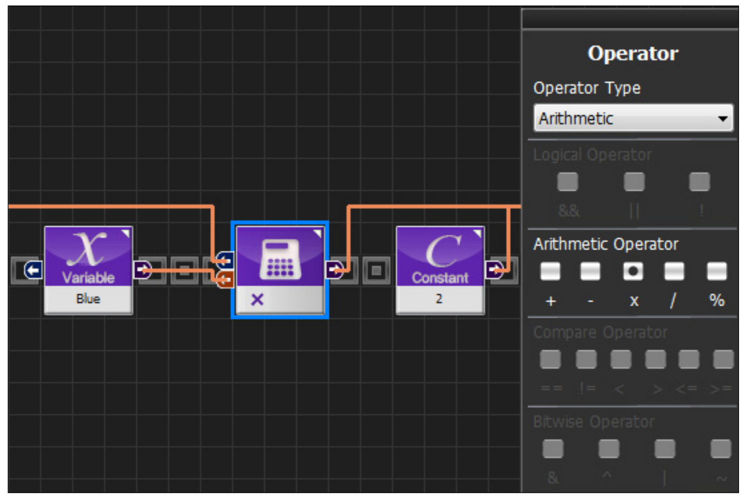
As explained previously, LED lights up according to the value is entered in the LED module. Since values in Red, Green, and Blue are saved as True (1), False (0), value entered in the LED module has be in the following format (4 x Blue + 2 x Green + 1 x Red).


Variable
Type
bool
Variable Name
Blue
Assign Constant Value
True False
Constant Value
True False

45 Read Blue Value

Add Blue variable to form required formula (4 x Blue + 2 x Green + 1 x Red)

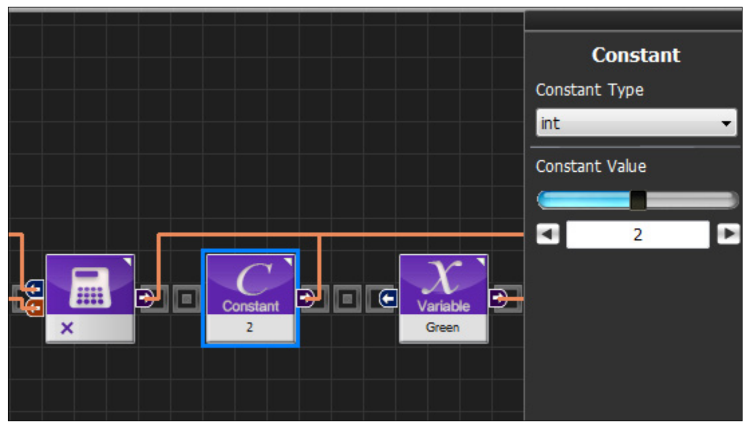
Select Data > Variable and place the module after the previous module.
Set Type : bool.
Enter Variable Name : Blue.


Operator
Operator Type
Arithmetic
Logical Operator
&& || !
Arithmetic Operator
+ - x / %
Compare Operator
== != < > <= >=
Bitwise Operator
& ^ | ~

46 Multiplication

Apply multiplication operator to multiply 4 by Blue value.

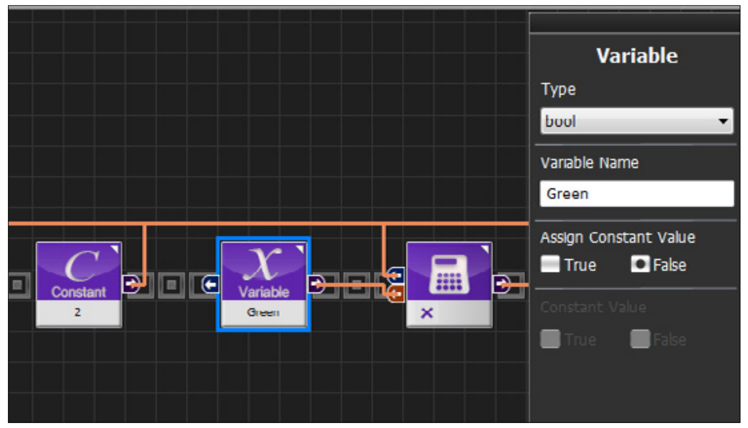
Select Data > Operator
Operator Type : Arithmetic
Set Logical Operator : X (Multiplication).
Connect output pin from 4 in step 44 and Blue from step 45 to the multiplication module.


Constant
Constant Type
int
Constant Value
2

47 Add Constant 2

Add Constant 2 to form formula (4 x Blue + 2 x Green + 1 x Red).

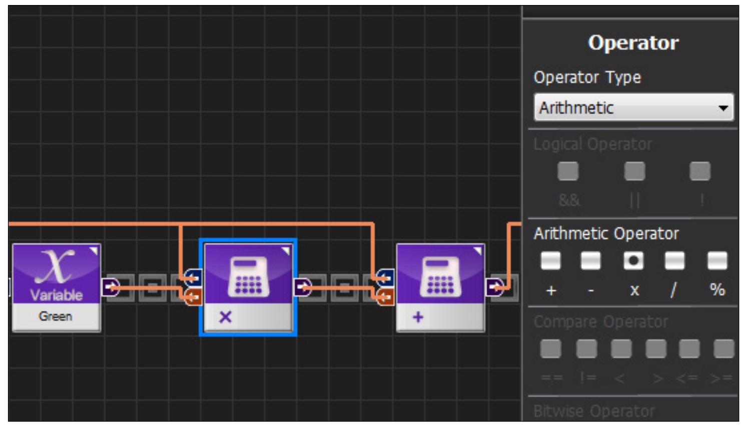
Select Data > Constant and place the module after the previous module.
Set Constant Type : int
Set Constant Value : 2


Variable
Type
bool
Variable Name
Green
Assign Constant Value
True False
Constant Value
True False

48 Read Green Value

Add Green variable to form required formula (4 x Blue + 2 x Green + 1 x Red)

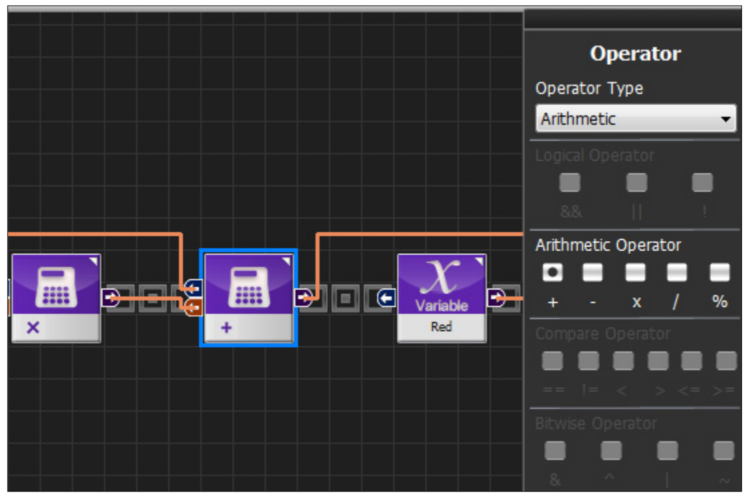
Select Data > Variable and place the module after the previous module.
Set Type : bool.
Enter Name : Green.


Operator
Operator Type
Arithmetic
Logical Operator
&& || !
Arithmetic Operator
+ - x / %
Compare Operator
== != < > <= >=
Bitwise Operator

49 Multiplication

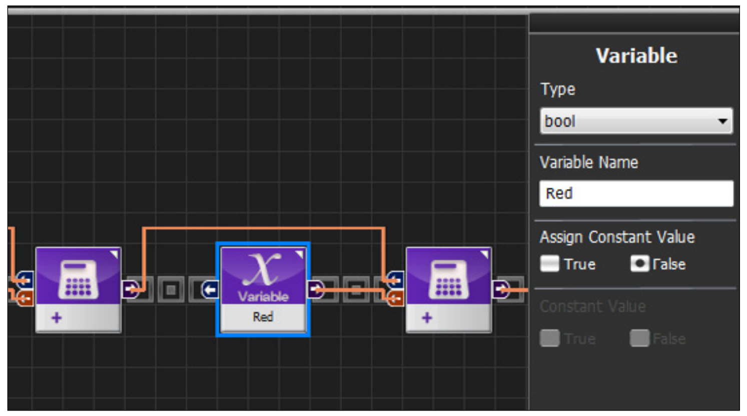
Apply multiplication operator to multiply 2 by Green value.

Select Data > Operator
Operator Type : Arithmetic
Set Logical Operator : X (Multiplication).
Connect output pin from 2 in step 47 and Green from step 48 to the multiplication module.



50 Addition

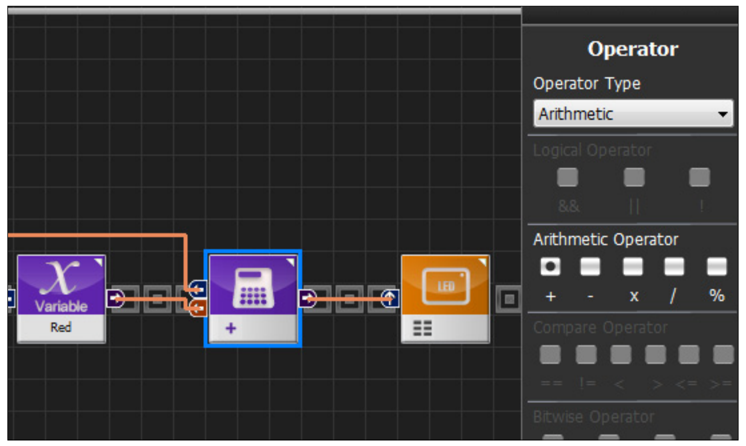
Apply Addition operator to add 4 x Blue to 2 x Green.
Select Data > Operator
Set Operator Type : Arithmetic
Select Logical Operator : +(Addition).
Connect output pins from Multiplication modules in step 46 and 49 to the Addition module.



51 Read Red Value

Add the last item of the formula (4 x Blue + 2 x Green + 1 x Red) Red variable. 1 does not need to be added.

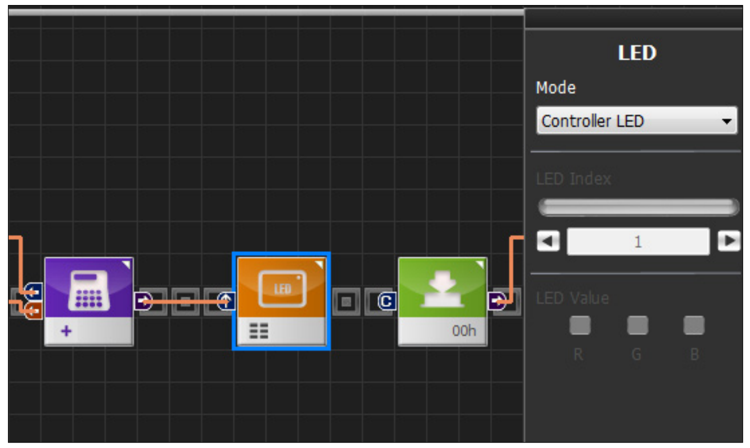
Select Data > Variable and place the module after the previous module.
Set Type : bool
Enter Variable Name : Red.



52 Addition

Apply Addition operator to add 4 x Blue + 2 x Green to Red.

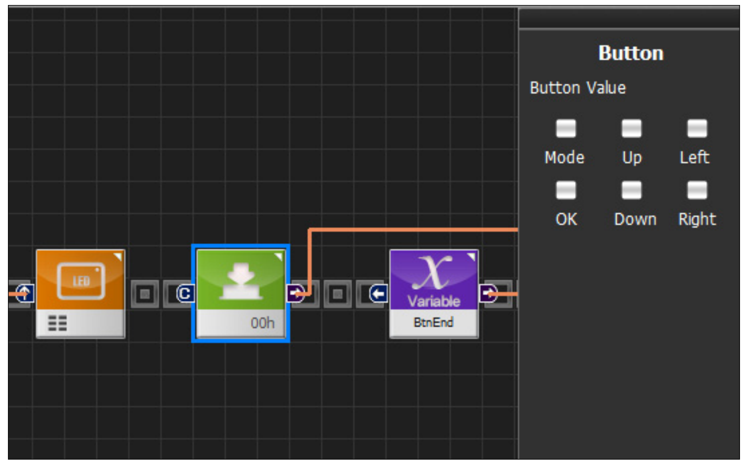
Select Data > Operator
Set Operator Type : Arithmetic
Select Logical Operator : +(Addition).
Connect output pins from Addition modules in step 50 and output pin from Red outtput in step 51 to the Addition module.



53 Add LED Module

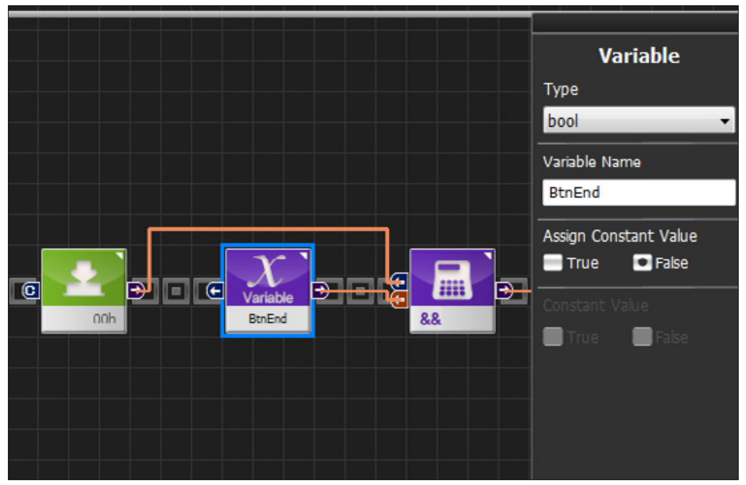
Add LED module and enter the value (4 x Blue + 2 x Green + 1 x Red).

Select Motion > LED and place the module after the previous module.
Set Mode : Controller LED
Connect the output pin from the Addition module in step 52 to the LED module input pin.



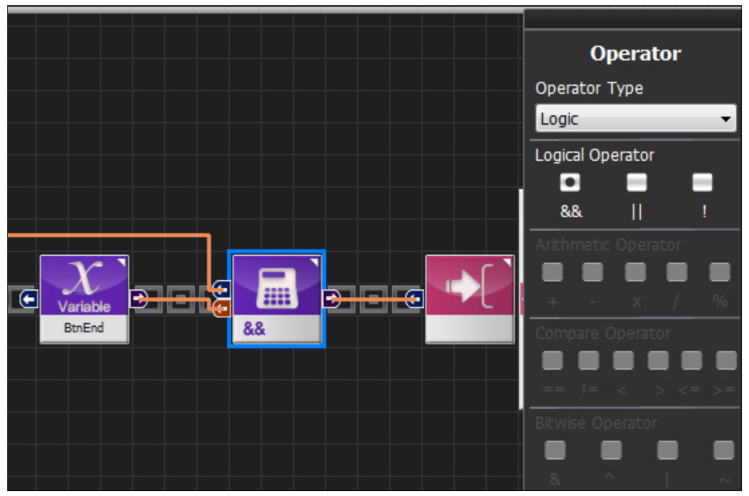
54 Detect Button Release

Variable value changed when button was pressed and BtnEnd value changed to True to prevent the variable value from changing again. However, BtnEnd value has to be changed to False once the button is released.



55 Add BtnEnd

Add module to read BtnEnd value.
Select Data > Variable and place the module after the previous module.
Set Type : bool
Enter Variable Name : BtenEnd.

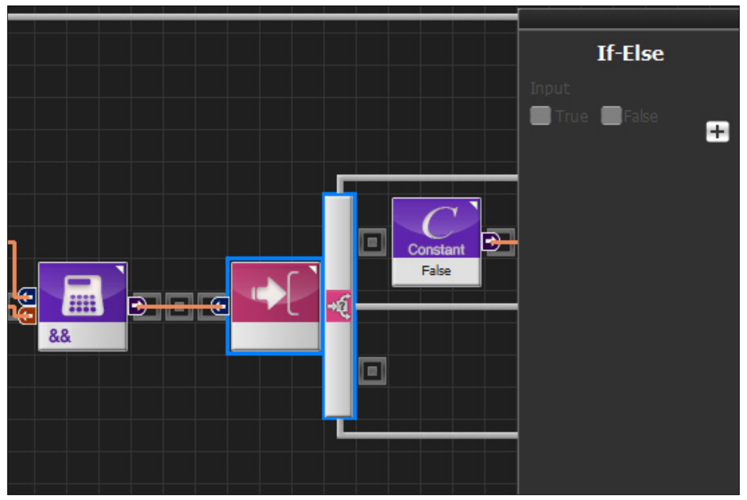


56 AND Operation

Apply AND operator to output 'True' when button is released and BtnEnd is 'True'.

Select Data > Operator
Set Operator Type : Logic
Set Logical Operator : &&(Logical AND).

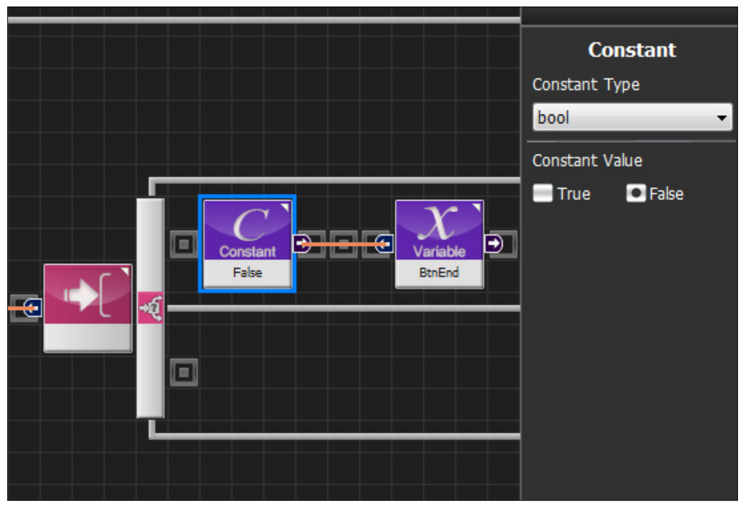
Connect output pin from the Button module in step 54 and BtnEnd module in step 55 to the AND module input pin.



57 Add If-Else Statement

If the value from AND module is 'True', add If-Else statement to turn BtnEnd to 'False'.

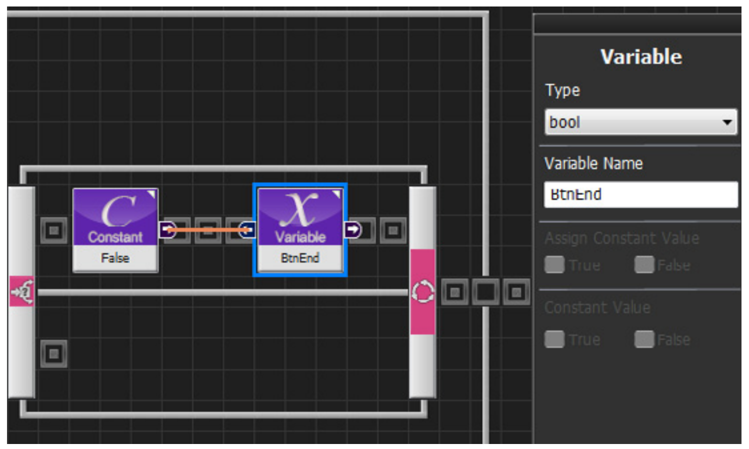
Select Flow > If-Else and place the module after the previous module.
Connect the AND module output pin from step 56 to the If-Else module input pin.



58 Add Constant

Add 'False' Constant to turn BtnEnd to 'False'.

Select Data > Constant and place the module in the highest program line of the If-Else module.

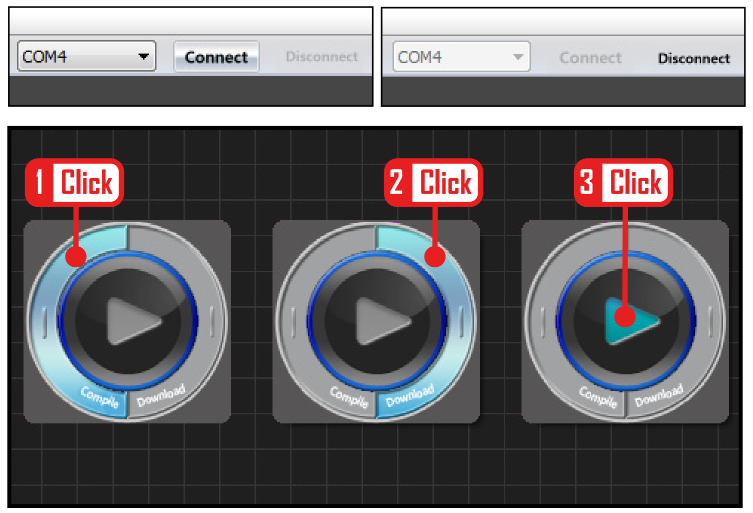


59 Substitue 'False' In BtnEnd

Add BtnEnd variable and substitute 'False' as the value.

Select Data Variable and place the module after the previous module.

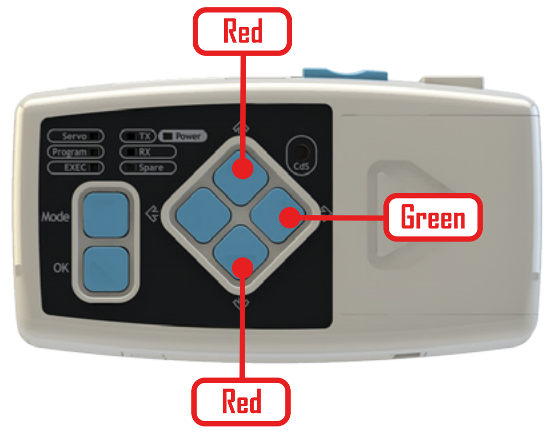
Set Type : bool.
Enter Variable Name : BtnEnd.
Connect the output pin from 'False' in step 58 to BtnEnd input pin.



60 Download

After programming is completed.
Compile -> Download -> Run.

Select the appropriate COMPORT or USB-to-Serial number located on the PC and press Connect to open the serial port.
Click 'Compile' and then click 'Download' on the right to download the program to the robot if no compile error is found.
Click "Run" button (arrow in the middle) after completing the download.



61 Robot Motion

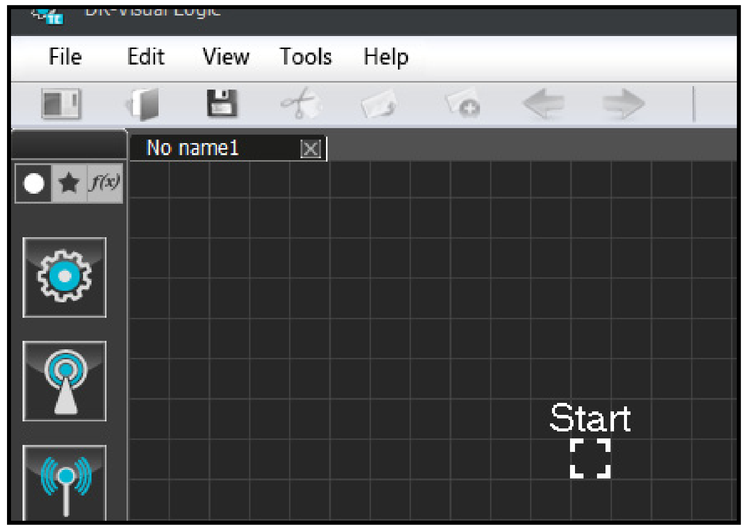
Press Up button for Red
Press Right button for Green
Press Down button for Blue
LED will turn on and go off when button is pressed once more.

Programming Individual Module
Light Sensor 08-4

Example Description

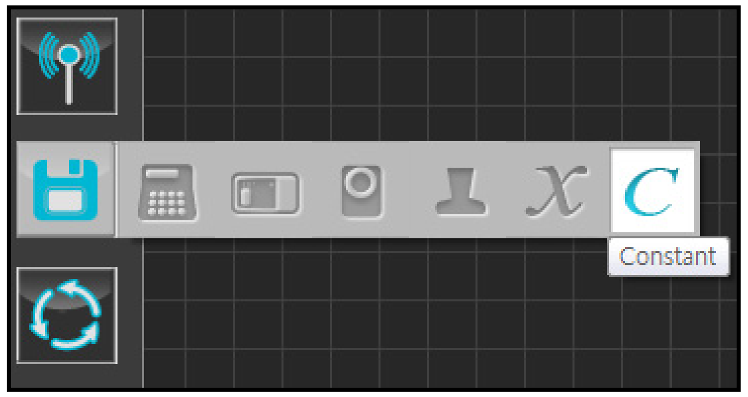
Program will operate the robot motor according to the amount of light being detected.

Robot will lift up the right arm when environment becomes dark (Cover the CDS sensor located at back of the controller to decrease amount of light being detected and robot will respond by lifting up the right hand).



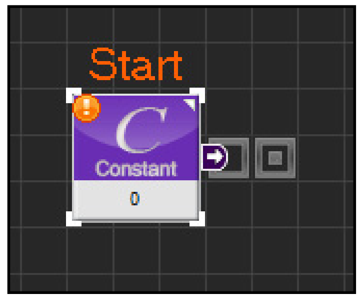
01 Create New File

Click on the left most icon in the tools to create new dts file. Select HOVIS Lite/Eco as the Robot.



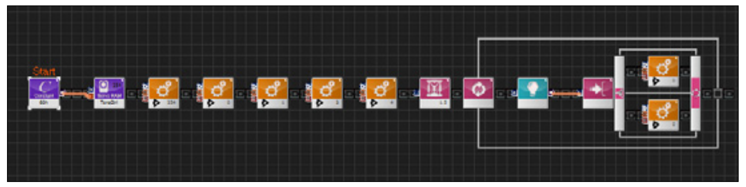
02 Select Module

Prior to placing the module, click to select the module from the module bar located on the left side.
Click Data > Constant module



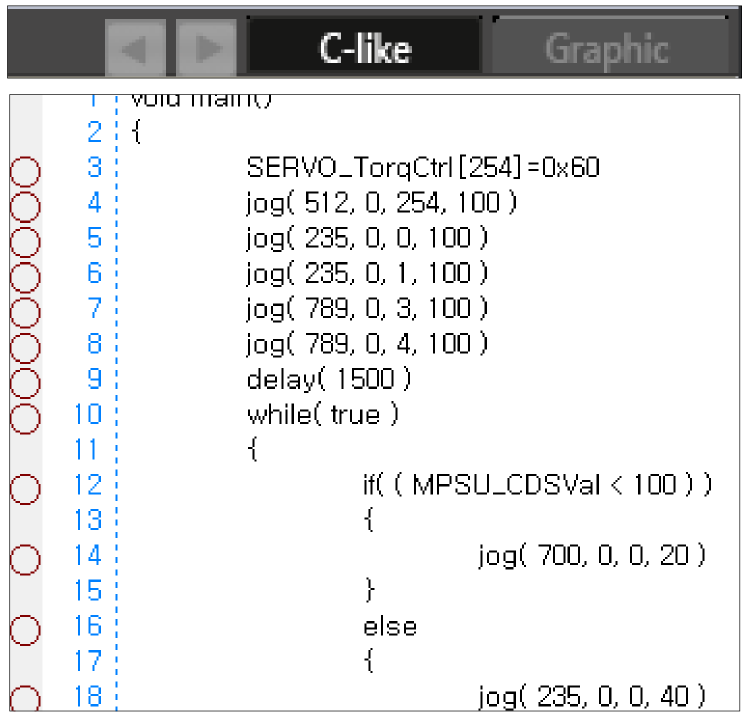
03 Place Module

Drag and dock the module to the Start Point to activate the module. Active module will turn to color



04 Entire Program

Entire program using light sensor to control the robot motors.

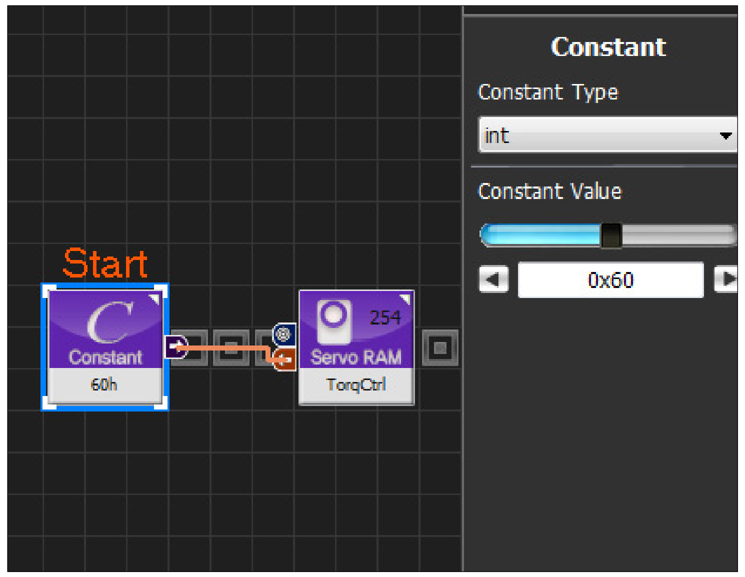


```
void main()
{
        SERVO_TorqCtrl[254]=0x60
        jog( 512, 0, 254, 100 )
        jog( 235, 0, 0, 100 )
        jog( 235, 0, 1, 100 )
        jog( 789, 0, 3, 100 )
        jog( 789, 0, 4, 100 )
        delay( 1500 )
        while( true )
        {
                if( ( MPSU_CDSVal < 100 ) )
                {
                        jog( 700, 0, 0, 20 )
                }
                else
                {
                        jog( 235, 0, 0, 40 )
```

05 Viewing C-Like

Click "C-Like" tab near the top right to open the program source code window as shown in the left.

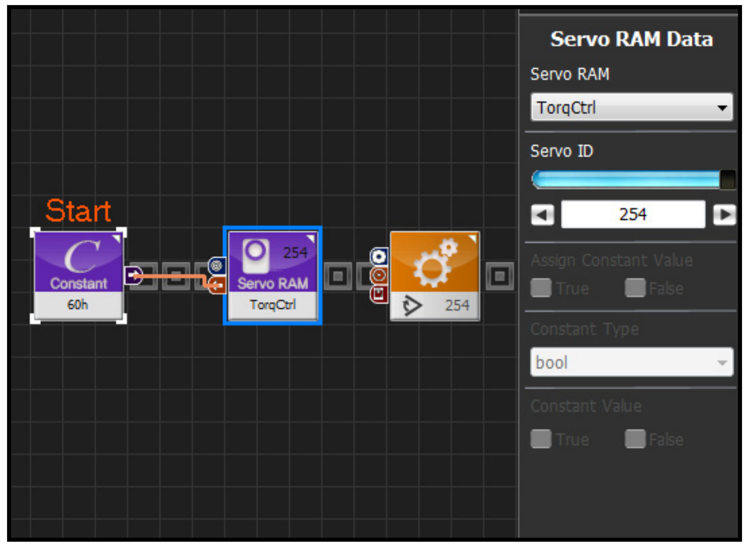
Since codes are very similar to the C language structure, studying the codes will help the user become familiar with the C language as well. Clicking on the module results in cursor jumping to the matching source code line in the C-Like window. This feature enables the user to easily see the codes generated by the module.



Constant
Constant Type: int
Constant Value: 0x60

06 Set Constant

This section allows the servo motor to operate on its own. From the Constant module properties, click on the Constant Value and change the value to 0x60. Constant with Hex number of 0x60 shows the torque enabled state of the servo motor. This value is sent to the input pin of the next module through the output pin.

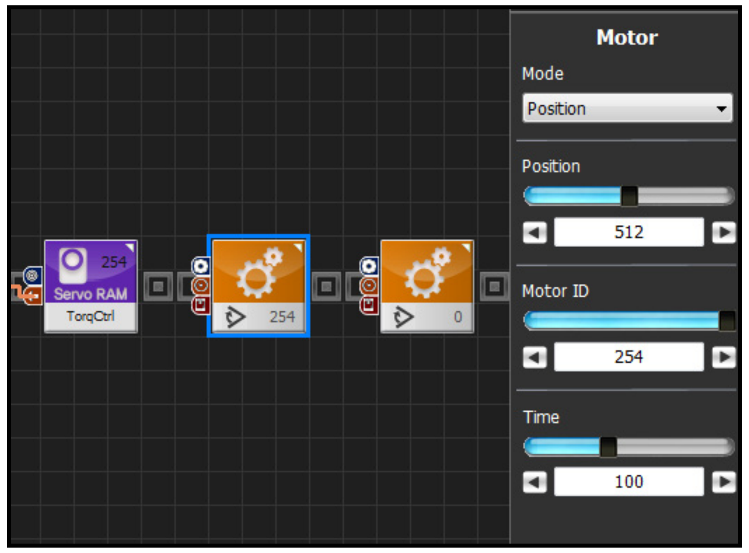


Servo RAM Data
Servo RAM: TorqCtrl
Servo ID: 254
Assign Constant Value: True False
Constant Type: bool
Constant Value: True False

07 Apply to All servos

This section applies constant value 0x60 received from the previous section to all servo motors.

Select Data > Servo RAM and place it after the Constant module.
Servo RAM : Select TorqCtrl
Servo ID : Set to 254, ID 254 means setup will be applied to all connected servos.
Use the Connector to connect the output pin of the Constant module to the second input pin of the Servo RAM module.

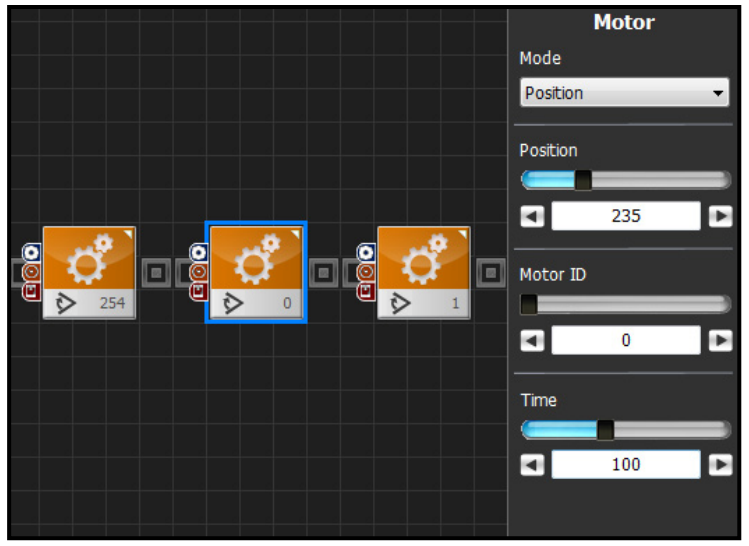


Motor
Mode: Position
Position: 512
Motor ID: 254
Time: 100

08 Set Position to All Servos

This section sets all servo motor positions to the center.
Select Motion > Motor and place it after the previous module.

Set Mode : Position, to control the angle.
Set Position : 512, Position value of 512 will send the motor to the 0 point (center).
Set Motor ID : 254, ID 254 will apply the setup to all connected servo motors.
Set Time : 100, 1 unit = 11.2ms, value of 100 is approximately 1.12s. Motors will be sent to the desired position in 1.12s.



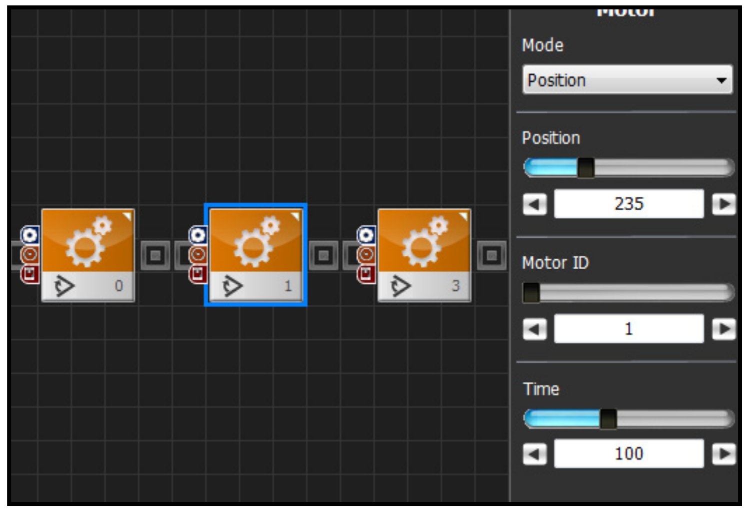
Motor
Mode: Position
Position: 235
Motor ID: 0
Time: 100

09 Moto ID 0 (Right Shoulder) Setup

When all servo motors are aligned to the center, humanoid robot will be standing with both arms stretched out to the side. In order to make applying motion easier, robot posture should be returned to the basic attention posture with both arms lowered to the side of the body.

Select Motion > Motor and place it after the previous module.

Set Mode : Position, to control the angle.
Set Position : 253. Changing the motor position to 253 will allow the horizontally stretched arm to be lowered to vertical position.
Set Motor ID : 0, right shoulder motor has ID 0.
Set Time : 100, motor will move to the required position in approximately 1.12s.

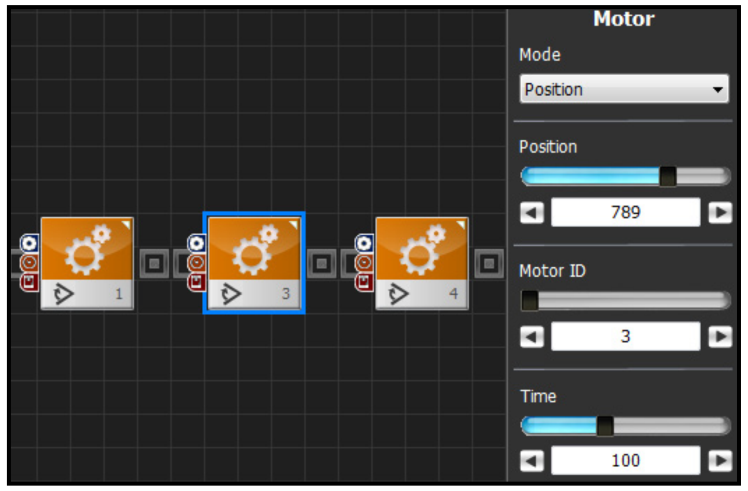


Motor
Mode: Position
Position: 235
Motor ID: 1
Time: 100

10 Set Motor ID 1 (Upper Right Arm)

Select Motion > Motor module and place it after the previous module.

Set Mode : Position to control the motor position.
Set Position : 235. Changing the motor position to 235 will lower the horizontally stretched arm to the new vertical position.
Set Motor ID : 1, upper right arm motor has ID 1.
Set Time : 100, motor will move to the required position in approximately 1.12s.

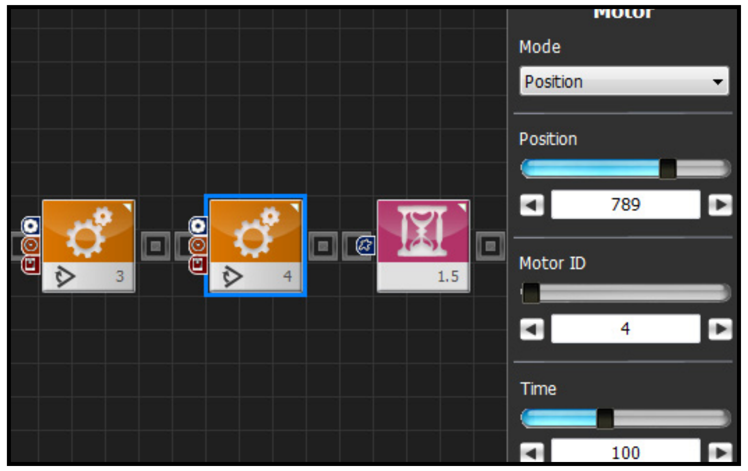


Motor
Mode: Position
Position: 789
Motor ID: 3
Time: 100

11 Set Motor ID 3 (Left Shoulder)

This section turns the left shoulder in order to lower the arm to the side of the body.
Select Motion > Motor module and place it after the previous module

Set Mode : Position to control the motor position.
Set Position : 789. Changing the motor position to 789 will allow the horizontally stretched arm to be lowered to vertical position.
Set Motor ID : 3 , left shoulder motor has ID 0.
Set Time : 100, motor will move to the required position in approximately 1.12s.

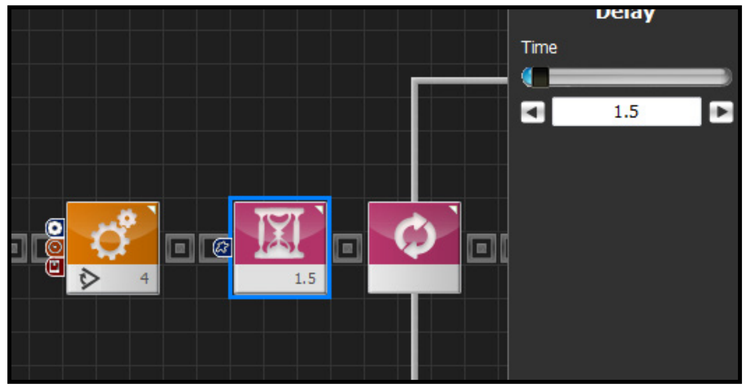


Motor
Mode: Position
Position: 789
Motor ID: 4
Time: 100

12 Set Motor ID 4 (Upper Left Arm)

Select Motion > Motor module and place it after the previous module.

Set Mode : Position to control the motor position.
Set Position : 789. Changing the motor position to 789 will lower the horizontally stretched arm to the new vertical position.
Set Motor ID : 4, upper left arm motor has ID 4.
Set Time : 100, motor will move to the required position in approximately 1.12s.



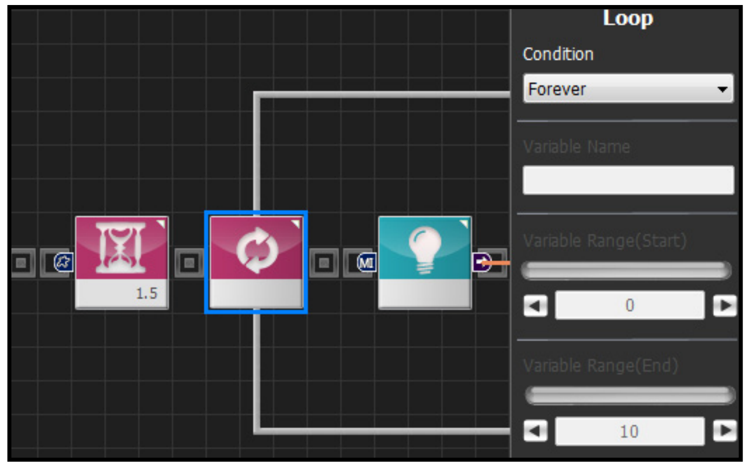
Delay
Time
1.5

13 Delay

This section adds 1.5s of delay before starting the next motor operation. Delay allows the motor to move to the position requested by the previous module without any interruption. Individual motion command to the motors 0 1 3 4 were sent immediately after the command to send all motors (254) to position 512. From users perspective, it would seem like robot lowered both arms simultaneously to take attention posture.

Select Flow > Delay module and place it after the previous module.

Set Time : 1.5s, delay 1.5s.

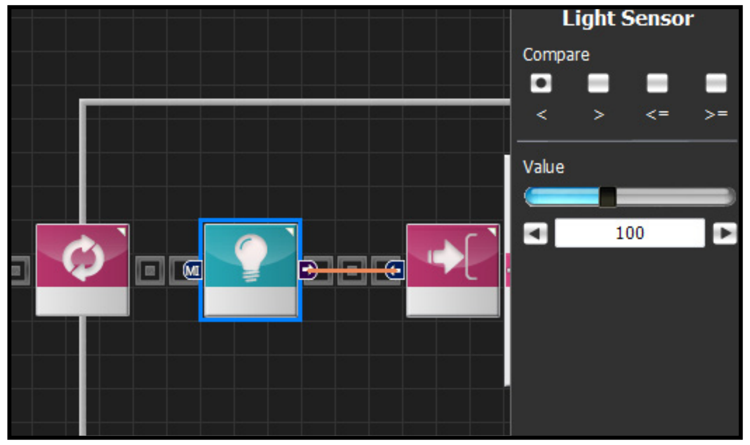


Loop
Condition
Forever
Variable Name
Variable Range(Start)
0
Variable Range(End)
10

14 Loop Statement

Insert loop statement to run the program in infinite loop.

Select Flow > Loop
Select Condition : Forever to create infinite loop statement without any condition.



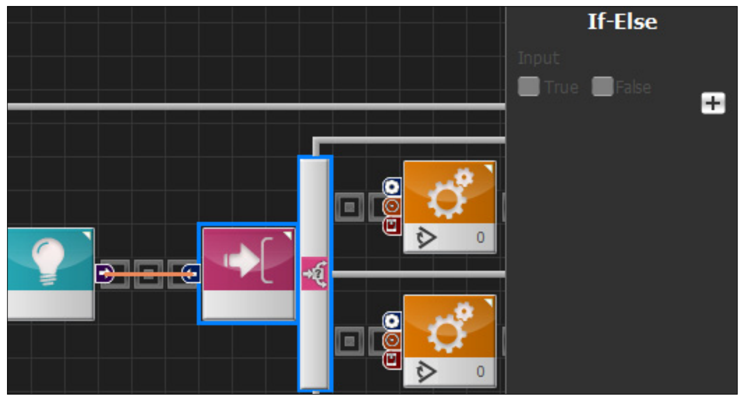
Light Sensor
Compare
< > <= >=
Value
100

15 Add Light Sensor

Add light sensor module. This module will output True/False after comparing the luminosity of the detected light with the Value.

Select Sensor > Light and place it within the loop module.

Select Compare : < . Output 'True' if Sensor value is less than Value.
Set Value : 100. 100 is the standard of measure.
Output 'True' if sensor value is less than 100 and output 'False' if value is equal to or greater than 100.

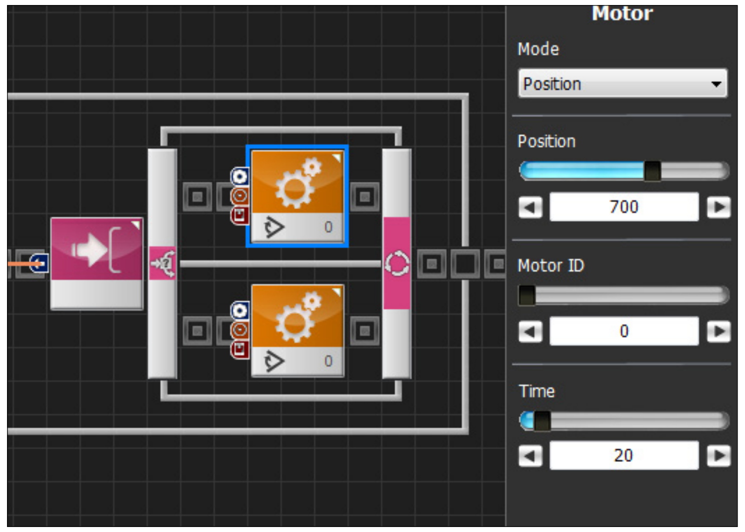


If-Else
Input
True False

16 If-Else Statement

Create conditional statement to produce different motion depending on the output of the light sensor.

Select Flow > If-Else and place the module after the previous module.
Connect the output pin from the light sensor in step 15 to the If-Else input pin.



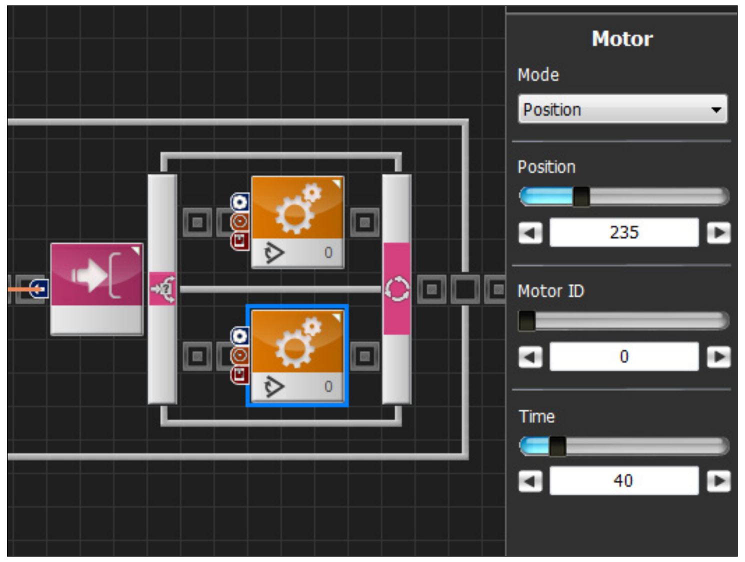
Motor
Mode
Position
Position
700
Motor ID
0
Time
20

17 Motor ID 0 (Right Shoulder) Setup

Lift up the right arm if the light sensor value is less than 100 (True). In other words, lift right arm if it's dark.

Select Motion > Motor and place it in the upper program line in the If-Else statement.

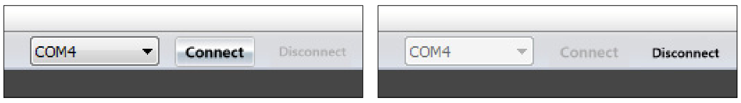
Set Mode : Position
Set Position : 700. Changing the motor position to 700 will lift the lowered arm straight forward.
Set Motor ID : 0, right shoulder motor has ID 0.
Set Time : 20

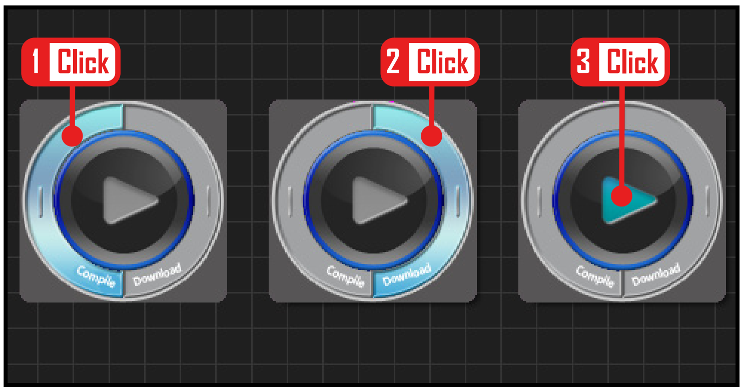


Motor
Mode
Position
Position
235
Motor ID
0
Time
40

18 Motor ID 0 (Right Shoulder) Setup

Lower the right arm if the light sensor value is equal to or greater than 100 (False). In other words, lower arm if it's light.
Select Motion > Motor and place it in the lower program line in the If-Else statement.
Set Mode : Position
Set Position : 235. Changing the motor position to 235 will lower the arm to the side of the body.
Set Motor ID : 0, right shoulder motor has ID 0.
Set Time : 40. Arm will lowered at a slightly slower pace than when it was being raised.

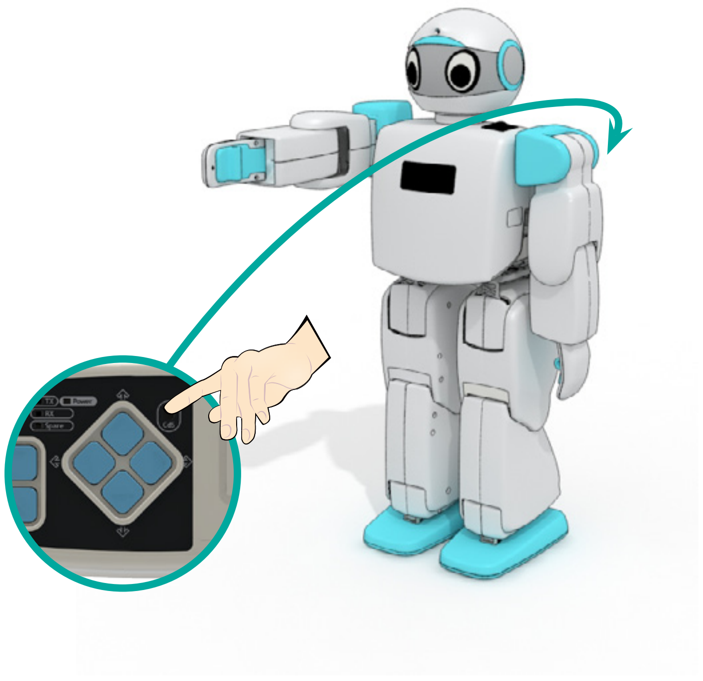




19 Download

After programming is completed.
Compile -> Download -> Run.

Select the appropriate COMPORT or USB-to-Serial number located on the PC and press Connect to open the serial port.
Click 'Compile' and then click 'Download' on the right to download the program to the robot if no compile error is found.
Click "Run" button (arrow in the middle) after completing the download.



20 Robot Motion

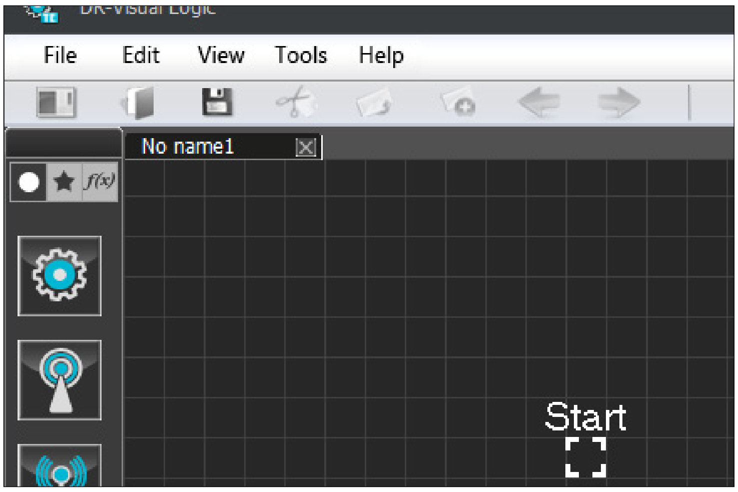
Robot is at attention posture under the bright light.
Robot will lift the right arm when the controller cds is covered.
Robot will lower the arm when the cds is uncovered.

08-5 Programming Individual Module Sound Sensor

Example Description.

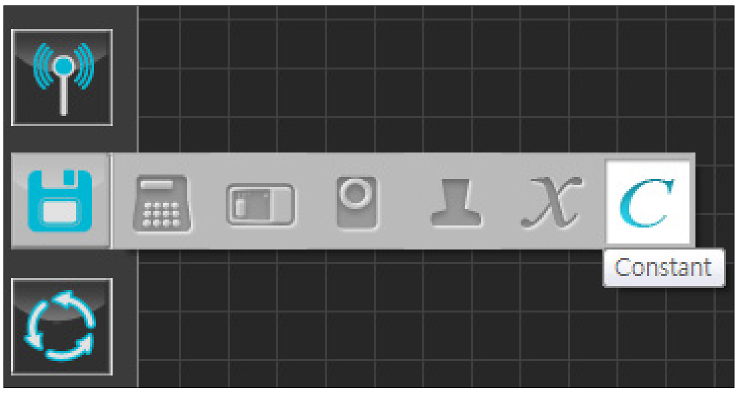
Sound sensors are located internally on both sides of the DRC controller. When loud sound such as hand clapping is detected, sound sensor module will compare the direction of the last detected sound to the Value and output either True or False.

Detected sound can have value range of -2 ~ +2. Value will be (-) if detected sound was from the left side and (+) if sound was from the right. Hand clap or finger snap from near the robot would work best for testing the program. If the sound source is too far, robot might have problem in detecting the sound direction due to the reflection from the wall or the floor, Robot will lift the left arm if sound detected is from the left and right arm is sound is from the right.



01 Create New File

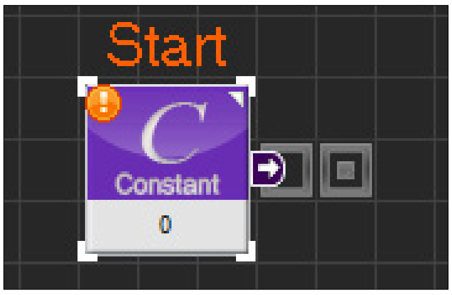
Click on the left most icon in the tools to create new dts file. Select HOVIS Lite/ Eco as the Robot.



02 Select Module

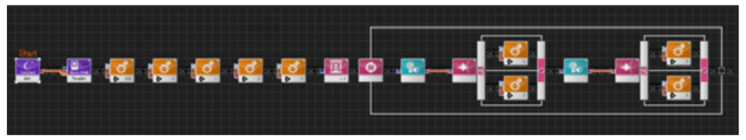
Prior to placing the module, click to select the module from the module bar located on the left side.

Click Data > Constant module



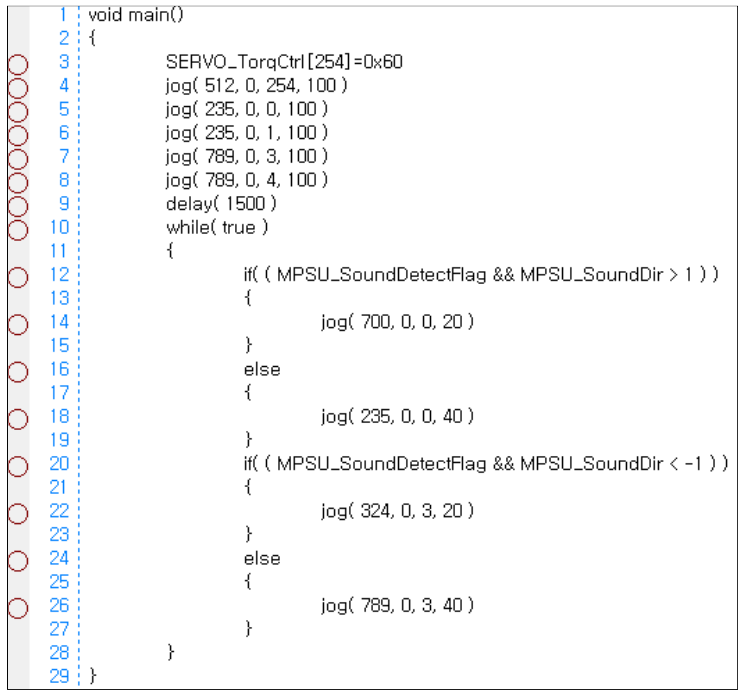
03 Place Module

Drag and dock the module to the Start Point to activate the module. Active module will turn to color.



04 Entire Program

View of the entire program.

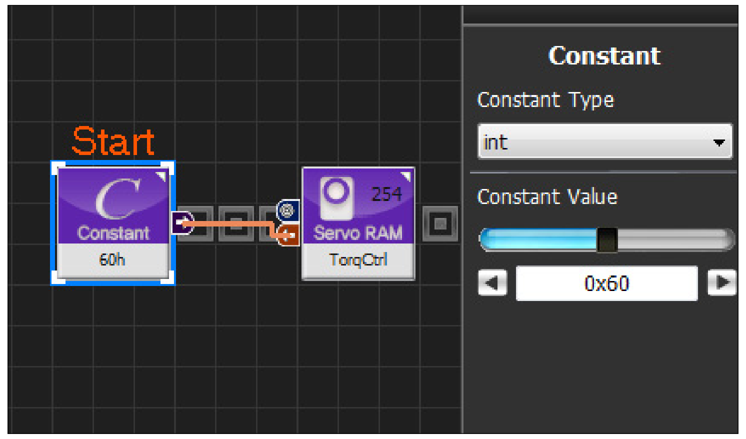


```
void main()
{
            SERVO_TorqCtrl[254]=0x60
            jog( 512, 0, 254, 100 )
            jog( 235, 0, 0, 100 )
            jog( 235, 0, 1, 100 )
            jog( 789, 0, 3, 100 )
            jog( 789, 0, 4, 100 )
            delay( 1500 )
            while( true )
            {
                        if( ( MPSU_SoundDetectFlag && MPSU_SoundDir > 1 ) )
                        {
                                    jog( 700, 0, 0, 20 )
                        }
                        else
                        {
                                    jog( 235, 0, 0, 40 )
                        }
                        if( ( MPSU_SoundDetectFlag && MPSU_SoundDir < -1 ) )
                        {
                                    jog( 324, 0, 3, 20 )
                        }
                        else
                        {
                                    jog( 789, 0, 3, 40 )
                        }
            }
}
```

05 Viewing C-Like

Click “C-Like” tab near the top right to open the program source code window as shown in the left.

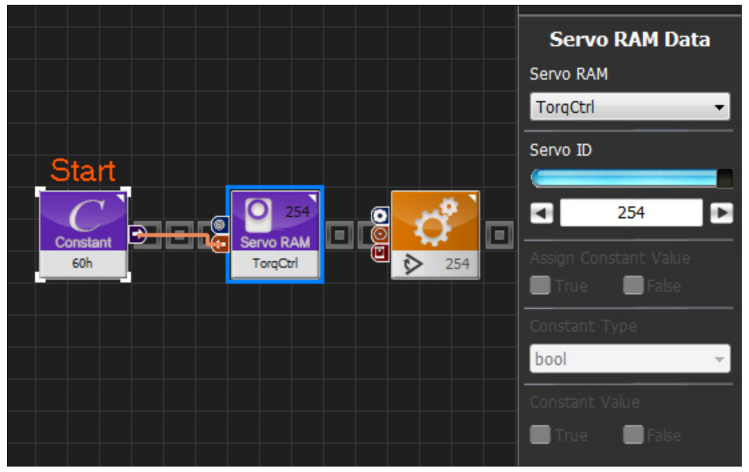
Since codes are very similar to the C language structure, studying the codes will help the user become familiar with the C language as well. Clicking on the module results in cursor jumping to the matching source code line in the C-Like window. This feature enables the user to easily see the codes generated by the module.



Constant
Constant Type: int
Constant Value: 0x60

06 Set Constant

This section allows the servo motor to operate on its own. From the Constant module properties, click on the Constant Value and change the value to 0x60. Constant with Hex number of 0x60 shows the torque enabled state of the servo motor. This value is sent to the input pin of the next module through the output pin.

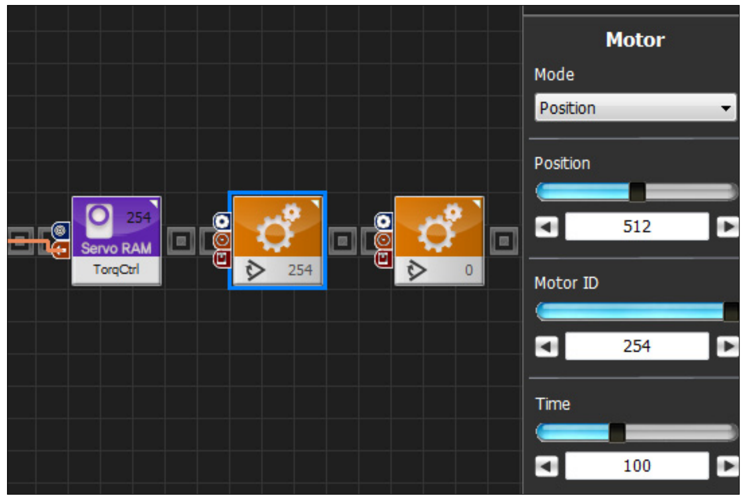


Servo RAM Data
Servo RAM: TorqCtrl
Servo ID: 254
Assign Constant Value: True False
Constant Type: bool
Constant Value: True False

07 Apply to All servos

This section applies constant value 0x60 received from the previous section to all servo motors.

Select Data > Servo RAM and place it after the Constant module.
Servo RAM : Select TorqCtrl
Servo ID : Set to 254, ID 254 means setup will be applied to all connected servos.
Use the Connector to connect the output pin of the Constant module to the second input pin of the Servo RAM module.

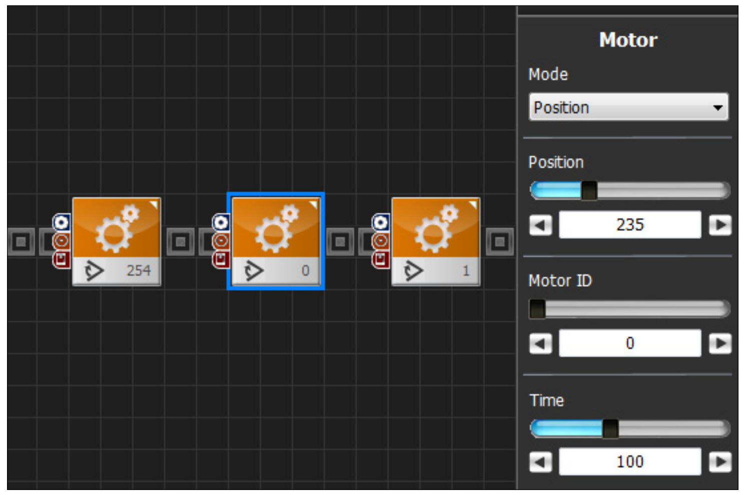


Motor
Mode: Position
Position: 512
Motor ID: 254
Time: 100

08 Set Position to All Servos

This section sets all servo motor positions to the center.

Select Motion > Motor and place it after the previous module.
Set Mode : Position, to control the angle.
Set Position : 512, Position value of 512 will send the motor to the 0 point (center).
Set Motor ID : 254, ID 254 will apply the setup to all connected servo motors.
Set Time : 100, 1 unit = 11.2ms, value of 100 is approximately 1.12s. Motors will be sent to the desired position in 1.12s.

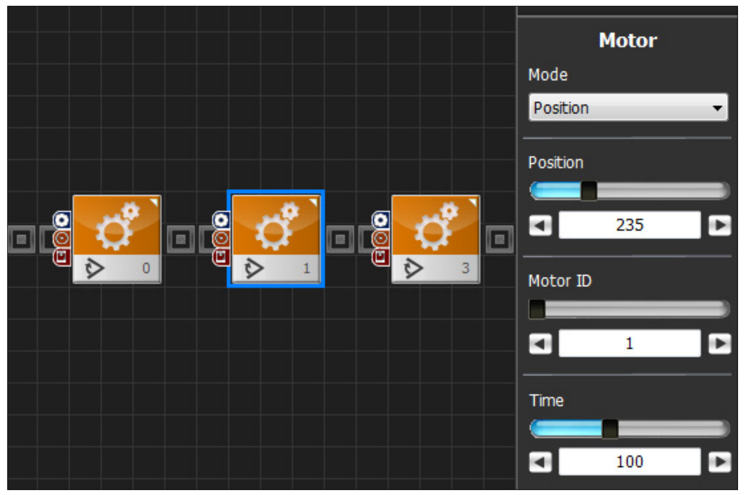


Motor
Mode: Position
Position: 235
Motor ID: 0
Time: 100

09 Moto ID 0 (Right Shoulder) Setup

When all servo motors are aligned to the center, humanoid robot will be standing with both arms stretched out to the side. In order to make applying motion easier, robot posture should be returned to the basic attention posture with both arms lowered to the side of the body.

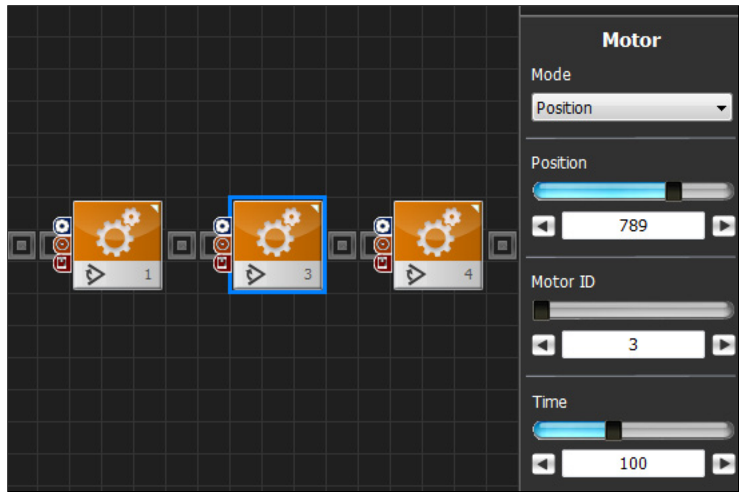
Select Motion > Motor and place it after the previous module.
Set Mode : Position, to control the angle.
Set Position : 235. Changing the motor position to 235 will allow the horizontally stretched arm to be lowered to vertical


Motor
Mode: Position
Position: 235
Motor ID: 1
Time: 100

10 Set Motor ID 1 (Upper Right Arm)

Select Motion > Motor module and place it after the previous module.

Set Mode : Position to control the motor position.
Set Position : 235. Changing the motor position to 235 will lower the horizontally stretched arm to the new vertical position.
Set Motor ID : 1, upper right arm motor has ID 1.
Set Time : 100, motor will move to the required position in approximately 1.12s.

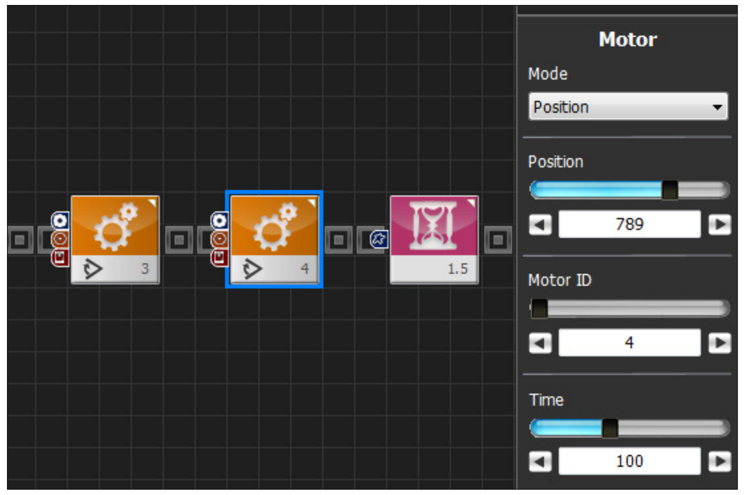

Motor
Mode: Position
Position: 789
Motor ID: 3
Time: 100

11 Set Motor ID 3 (Left Shoulder)

This section turns the left shoulder in order to lower the arm to the side of the body.

Select Motion > Motor module and place it after the previous module

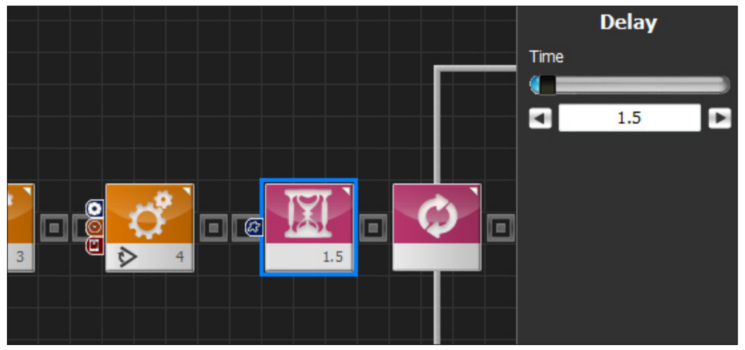
Set Mode : Position to control the motor position.
Set Position : 789. Changing the motor position to 789 will allow the horizontally stretched arm to be lowered to vertical position.
Set Motor ID : 3 , left shoulder motor has ID 0.
Set Time : 100, motor will move to the required position in approximately 1.12s.


Motor
Mode: Position
Position: 789
Motor ID: 4
Time: 100

12 Set Motor ID 4 (Upper Left Arm)

Select Motion > Motor module and place it after the previous module.

Set Mode : Position to control the motor position.
Set Position : 789. Changing the motor position to 789 will lower the horizontally stretched arm to the new vertical position.
Set Motor ID : 4, upper left arm motor has ID 4.
Set Time : 100, motor will move to the required position in approximately 1.12s.

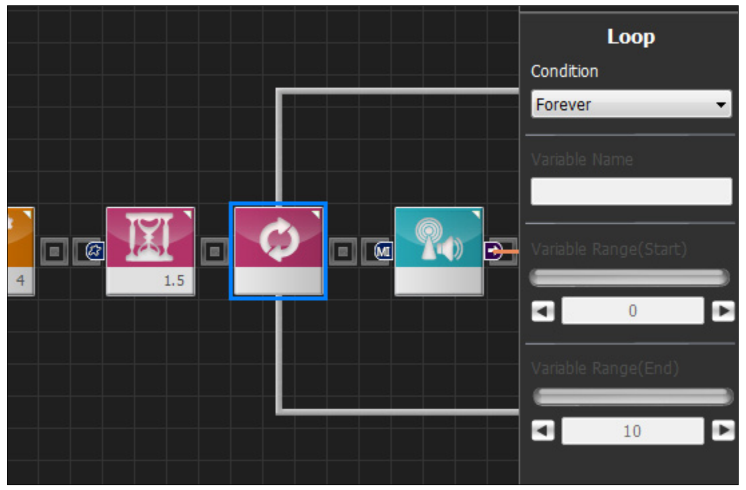

Delay
Time: 1.5

13 Delay

This section adds 1.5s of delay before starting the next motor operation. Delay allows the motor to move to the position requested by the previous module without any interruption. Individual motion command to the motors 0 1 3 4 were sent immediately after the command to send all motors (254) to position 512. From users perspective, it would seem like robot lowered both arms simultaneously to take attention posture.

Select Flow > Delay module and place it after the previous module.

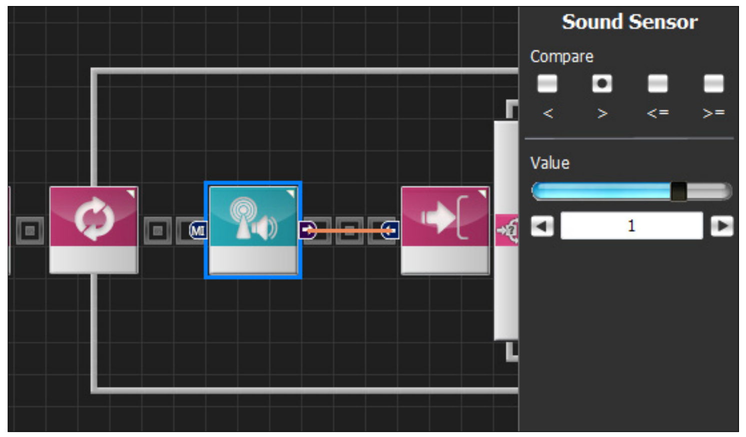
Set Time : 1.5s, delay 1.5s.


Loop
Condition: Forever
Variable Name
Variable Range(Start): 0
Variable Range(End): 10

14 Loop Statement

Insert loop statement to run the program in infinite loop.

Select Flow > Loop
Select Condition : Forever to create infinite loop statement without any condition.

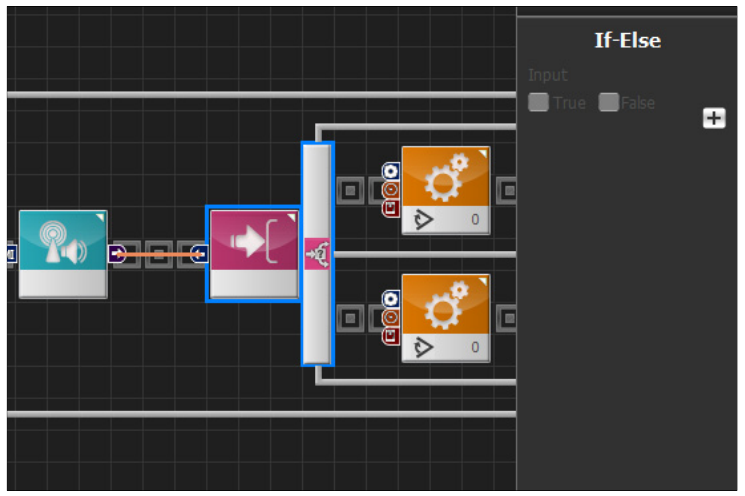

Sound Sensor
Compare: < > <= >=
Value: 1

15 Add Sound Sensor

Add Sound sensor module. This module will output True/False after comparing the direction of the detected sound with the Value.

Select Sensor > Sound and place it within the loop module.

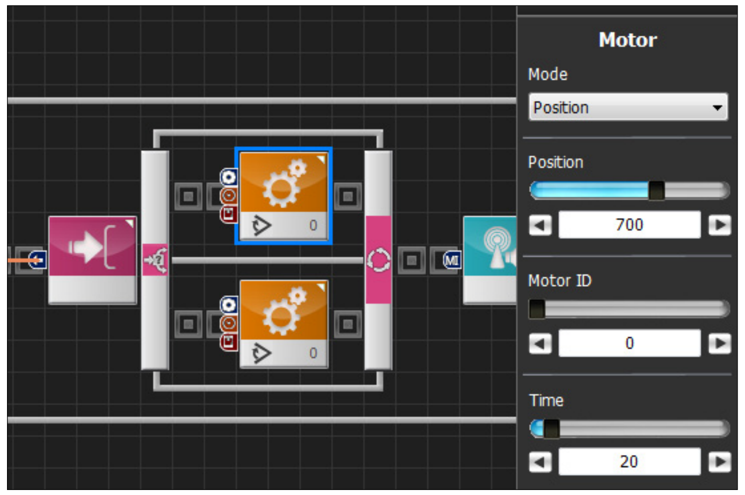
Select Compare : > . Output 'True' if Sensor value is greater than Value.
Set Value : 1. 1 means sound detected from the right side. Output 'True' if sensor value is greater than 1 and output 'False' if value is equal to or less than 1. In other words, value becomes 'True' if sound is from the right side.


If-Else
Input
True False

16 If-Else Statement

Create conditional statement to produce different motion depending on the output of the sound sensor.

Select Flow > If-Else and place the module after the previous module. Connect the output pin from the sound sensor in step 15 to the If-Else input pin.

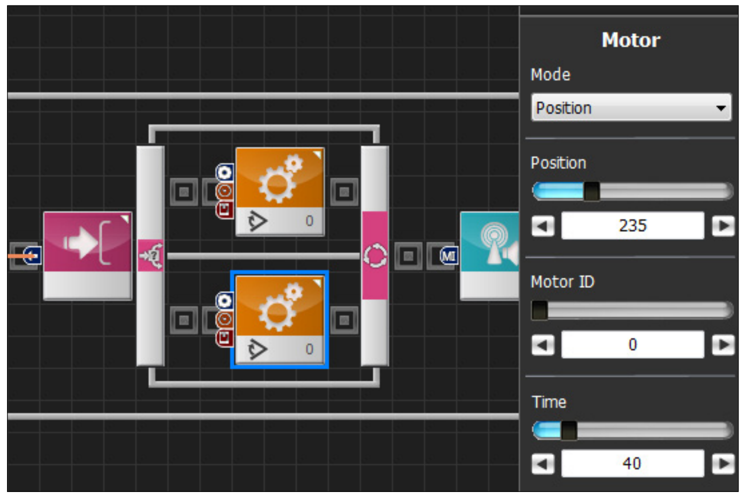

Motor
Mode: Position
Position: 700
Motor ID: 0
Time: 20

17 Motor ID 0 (Right Shoulder) Setup

Lift up the right arm if the sound sensor value is greater than 1 (True). In other words, lift right arm if sound was detected from the right side.

Select Motion > Motor and place it in the upper program line in the If-Else statement.

Set Mode : Position
Set Position : 700. Changing the motor position to 700 will lift the lowered arm straight forward.
Set Motor ID : 0, right shoulder motor has ID 0.
Set Time : 20

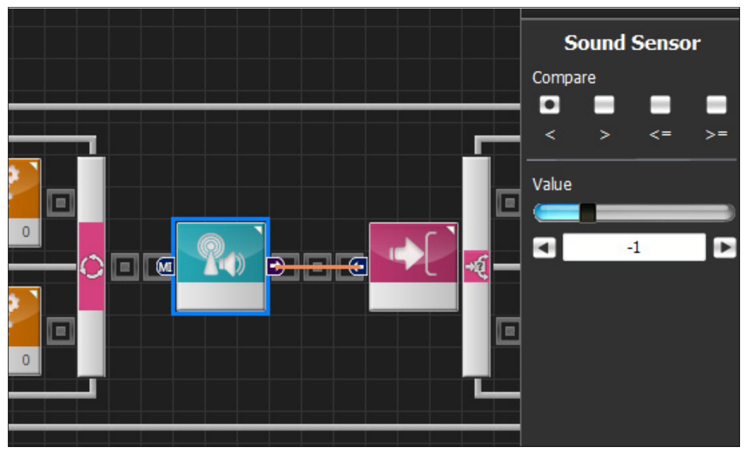

Motor
Mode: Position
Position: 235
Motor ID: 0
Time: 40

18 Motor ID 0 (Right Shoulder) Setup

Lower the right arm if no sound was detected or if the sound sensor value is equal to or less than 1 (False). In other words, lower arm if no sound is detected from the right side.

Select Motion > Motor and place it in the lower program line in the If-Else statement.

Set Mode : Position
Set Position : 235. Changing the motor position to 235 will lower the arm to the side of the body.
Set Motor ID : 0, right shoulder motor has ID 0.
Set Time : 40. Arm will be lowered at a slightly slower pace than when it was being raised.

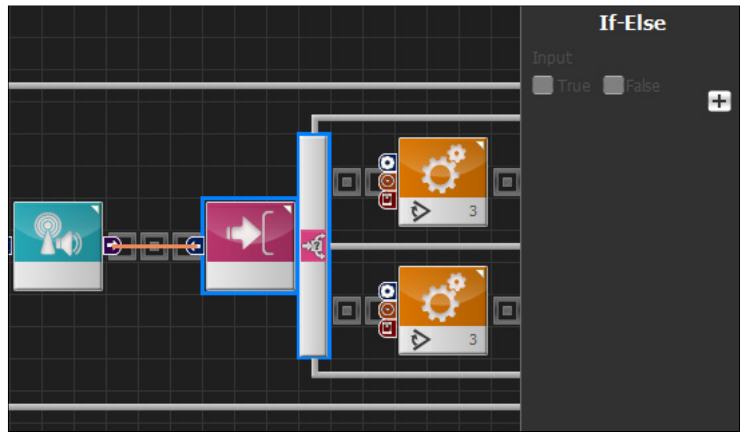

Sound Sensor
Compare
< > <= >=
Value: -1

19 Add Sound Sensor

Add Sound sensor module. This module will output True/False after comparing the direction of the detected sound with the Value.

Select Sensor > Sound and place it within the loop module.

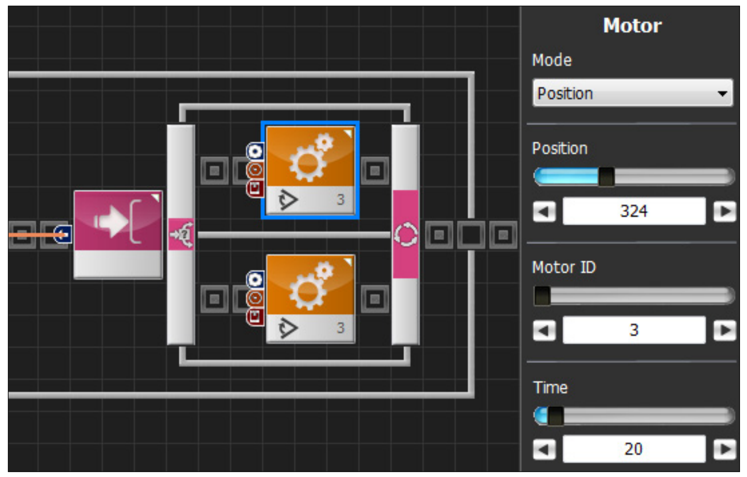
Select Compare : < . Output 'True' if Sensor value is less than Value.
Set Value : -1. -1 means sound detected from the left side. Output 'True' if sensor value is less than -1 and output 'False' if value is equal to or greater than -1. In other words, value becomes 'True' if sound is from the left side.


If-Else
Input
True False

20 If-Else Statement

Create conditional statement to produce different motion depending on the output of the sound sensor.

Select Flow > If-Else and place the module after the previous module. Connect the output pin from the sound sensor in step 19 to the If-Else input pin.

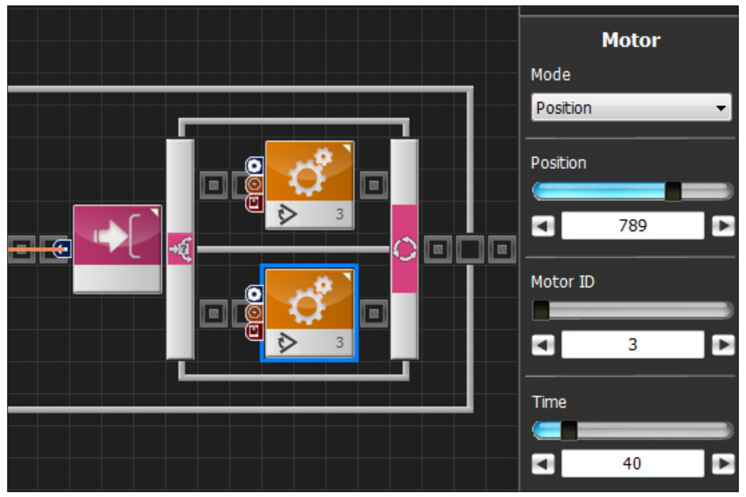

Motor
Mode: Position
Position: 324
Motor ID: 3
Time: 20

21 Motor ID 3 (Left Shoulder) Setup

Lift up the left arm if the sound sensor value is less than -1 (True). In other words, lift left arm if sound was detected from the left side.

Select Motion > Motor and place it in the upper program line in the If-Else statement.

Set Mode : Position
Set Position : 324. Changing the motor position to 324 will lift the lowered arm straight forward.
Set Motor ID : 3, left shoulder motor has ID 3.
Set Time : 20

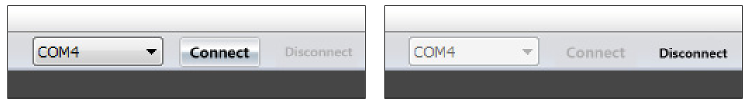


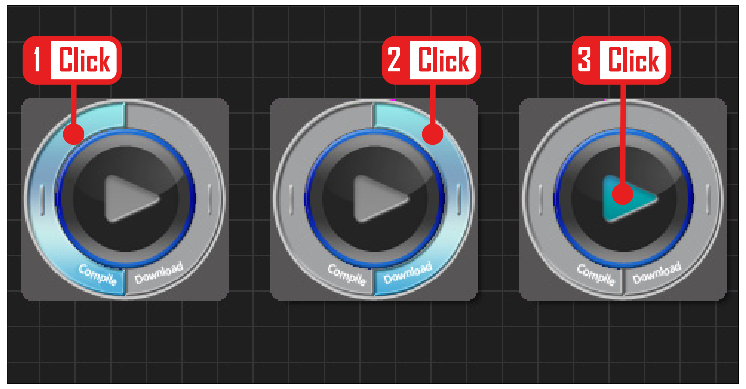
22 Motor ID 3 (Left Shoulder) Setup

Lower the left arm if no sound was detected or if the sound sensor value is equal to or greater than -1 (False). In other words, lower arm if no sound is detected from the left side.

Select Motion > Motor and place it in the lower program line in the If-Else statement.

Set Mode : Position
Set Position : 789. Changing the motor position to 789 will lower the arm to the side of the body.
Set Motor ID : 3, left shoulder motor has ID 0.
Set Time : 40. Arm will be lowered at a slightly slower pace than when it was being raised.

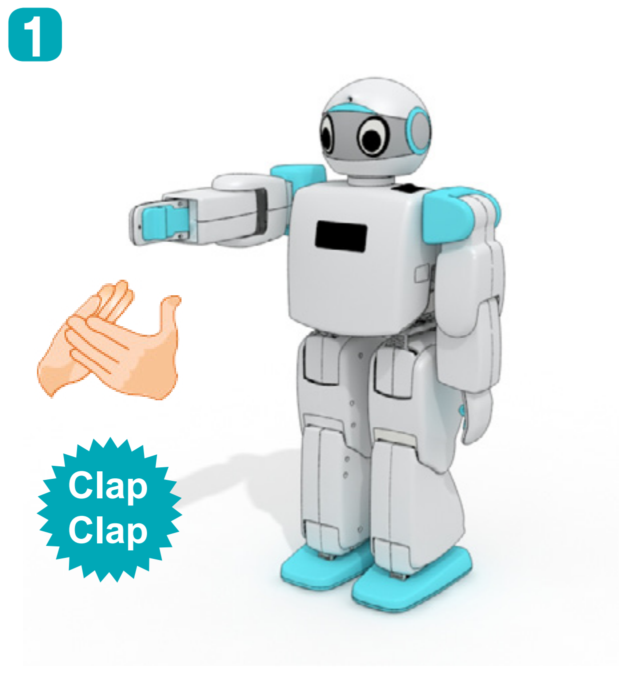


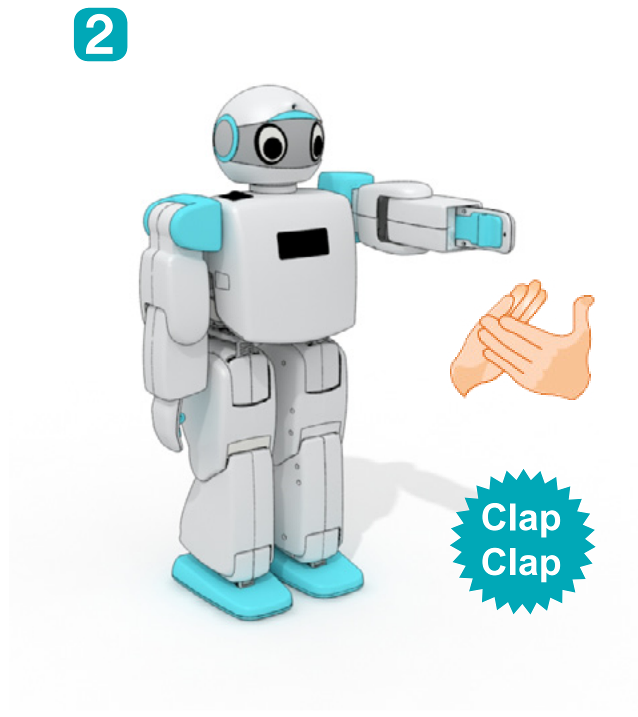


23 Download

After programming is completed.
Compile -> Download -> Run.

Select the appropriate COMPORT or USB-to-Serial number located on the PC and press Connect to open the serial port.
Click ‘Compile’ and then click ‘Download’ on the right to download the program to the robot if no compile error is found.
Click “Run” button (arrow in the middle) after completing the download.





24 Robot Motion

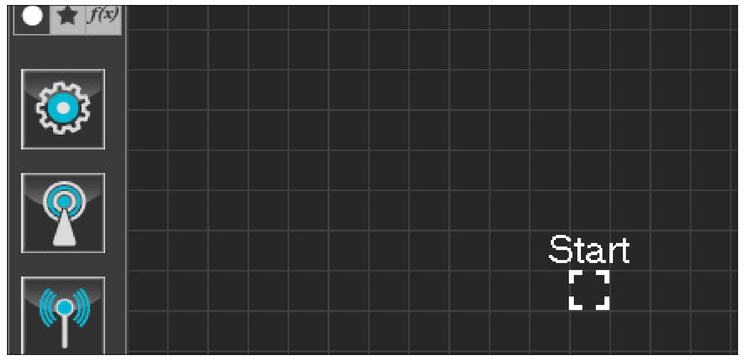
Robot will lift the left arm with left side clap and right arm with the right side clap.

Programming Individual Module

Sound Sensor (Improved) 08-6

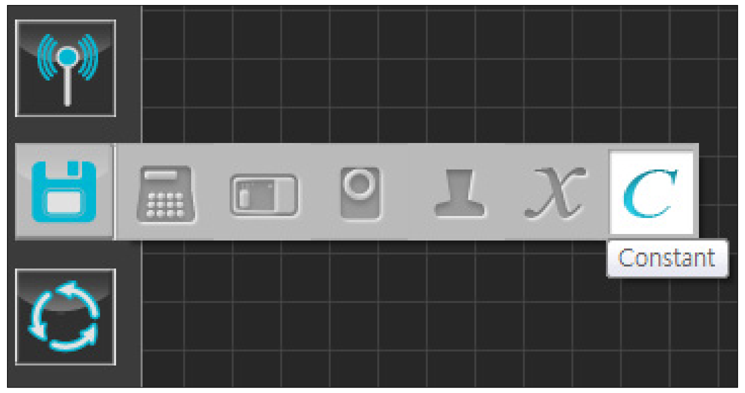
Example Description

Previous example described a program that lifted either the left hand or the right hand depending on the direction of the detected sound. However, program lacked accuracy due to interference from unexpected inputs from the surrounding noise or noise and vibration from the robot motors. One way to resolve this issue is to receive sound input only when the robot is standing still. This would prevent sound interference from the operating motors and allow detection of external sounds with relatively greater accuracy. No new sound input would be accepted while the robot is in motion and before sensor is reinitialized. 1s delay provides time for sound sensor to be reinitialized which occurs after 1s from the last sound input.



01 Create New File

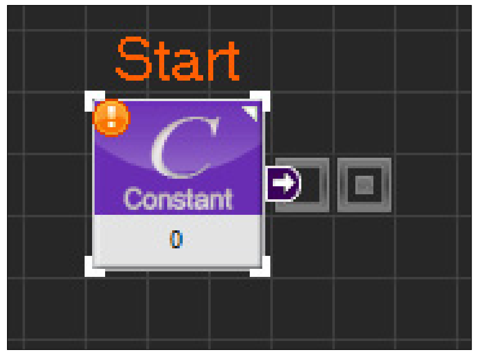
Click on the left most icon in the tools to create new dts file. Select HOVIS Lite/Eco as the Robot.



02 Select Module

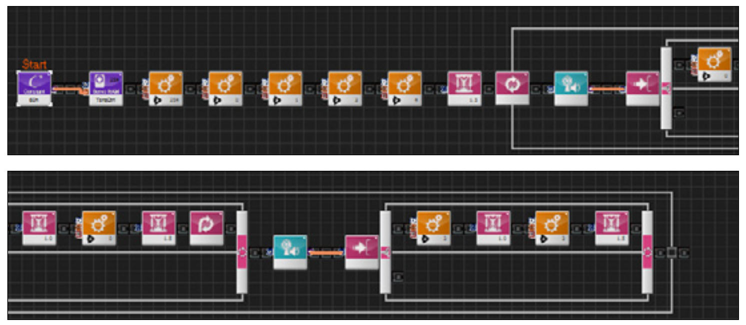
Prior to placing the module, click to select the module from the module bar located on the left side.

Click Data > Constant module.



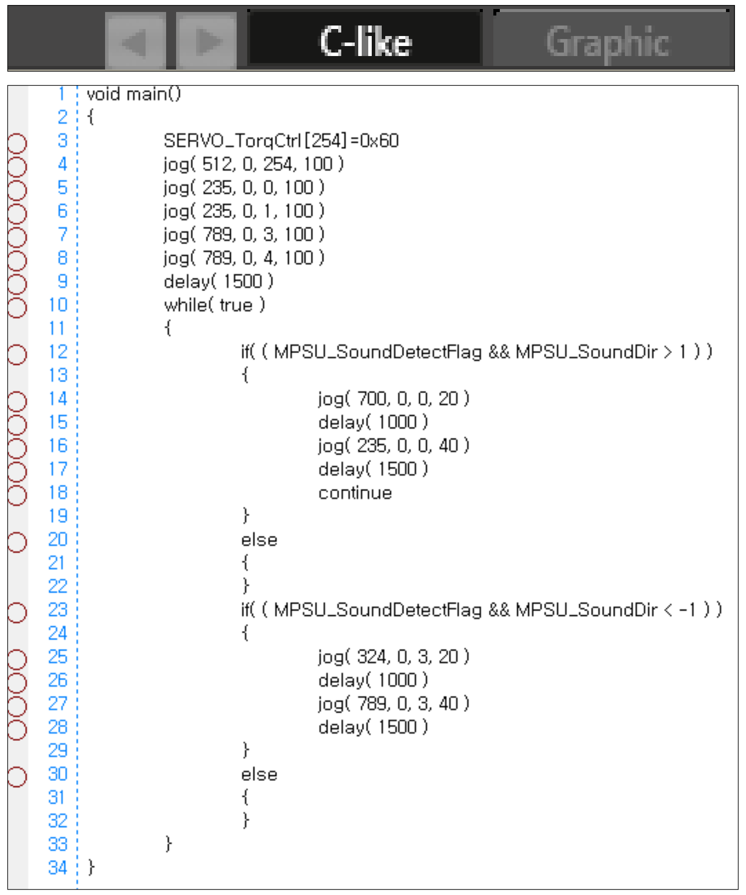
03 Place Module

Drag and dock the module to the Start Point to activate the module. Active module will turn to color.



04 Entire Program

View of the entire program.

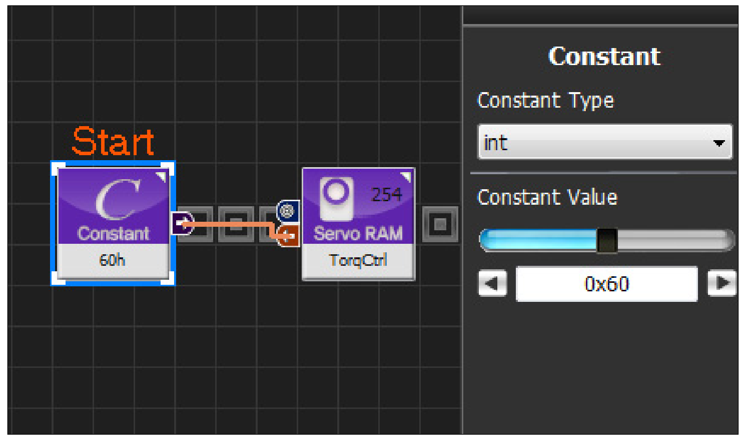


```
void main()
{
        SERVO_TorqCtrl[254]=0x60
        jog( 512, 0, 254, 100 )
        jog( 235, 0, 0, 100 )
        jog( 235, 0, 1, 100 )
        jog( 789, 0, 3, 100 )
        jog( 789, 0, 4, 100 )
        delay( 1500 )
        while( true )
        {
                if( ( MPSU_SoundDetectFlag && MPSU_SoundDir > 1 ) )
                {
                        jog( 700, 0, 0, 20 )
                        delay( 1000 )
                        jog( 235, 0, 0, 40 )
                        delay( 1500 )
                        continue
                }
                else
                {
                }
                if( ( MPSU_SoundDetectFlag && MPSU_SoundDir < -1 ) )
                {
                        jog( 324, 0, 3, 20 )
                        delay( 1000 )
                        jog( 789, 0, 3, 40 )
                        delay( 1500 )
                }
                else
                {
                }
        }
}
```

05 Viewing C-Like

Click "C-Like" tab near the top right to open the program source code window as shown in the left.

Since codes are very similar to the C language structure, studying the codes will help the user become familiar with the C language as well. Clicking on the module results in cursor jumping to the matching source code line in the C-Like window. This feature enables the user to easily see the codes generated by the module.



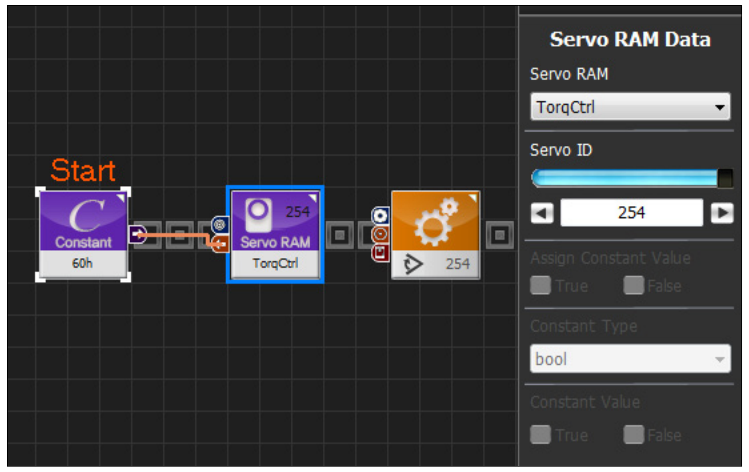
Constant

Constant Type: int

Constant Value: 0x60

06 Set Constant

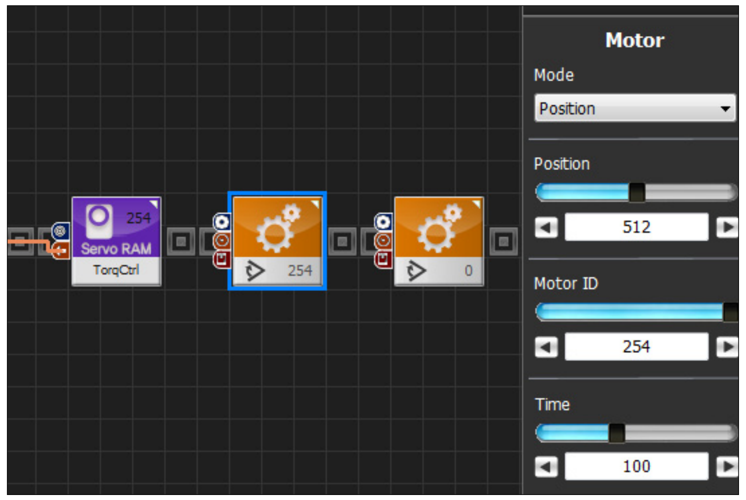
This section allows the servo motor to operate on its own. From the Constant module properties, click on the Constant Value and change the value to 0x60. Constant with Hex number of 0x60 shows the torque enabled state of the servo motor. This value is sent to the input pin of the next module through the output pin



Servo RAM Data
Servo RAM: TorqCtrl
Servo ID: 254
Assign Constant Value: True False
Constant Type: bool
Constant Value: True False

07 Apply to All servos

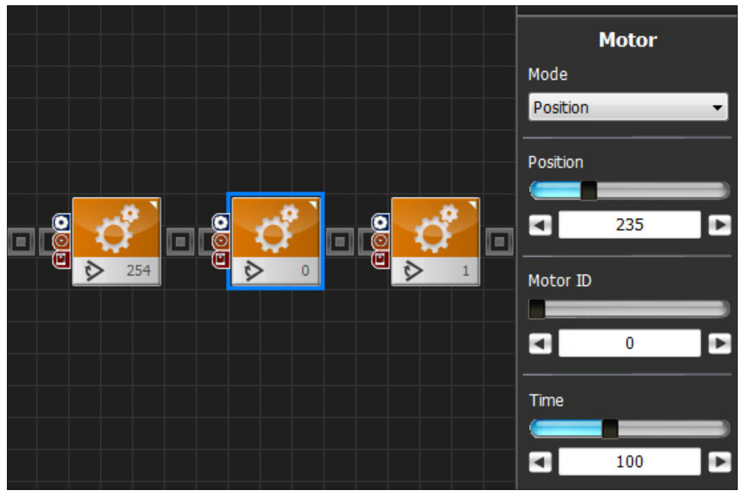
This section applies constant value 0x60 received from the previous section to all servo motors.
Select Data > Servo RAM and place it after the Constant module.
Servo RAM : Select TorqCtrl
Servo ID : Set to 254, ID 254 means setup will be applied to all connected servos.
Use the Connector to connect the output pin of the Constant module to the second input pin of the Servo RAM module.



Motor
Mode: Position
Position: 512
Motor ID: 254
Time: 100

08 Set Position to All Servos

This section sets all servo motor positions to the center.
Select Motion > Motor and place it after the previous module.
Set Mode : Position, to control the angle.
Set Position : 512, Position value of 512 will send the motor to the 0 point (center).
Set Motor ID : 254, ID 254 will apply the setup to all connected servo motors.
Set Time : 100, 1 unit = 11.2ms, value of 100 is approximately 1.12s. Motors will be sent to the desired position in 1.12s.

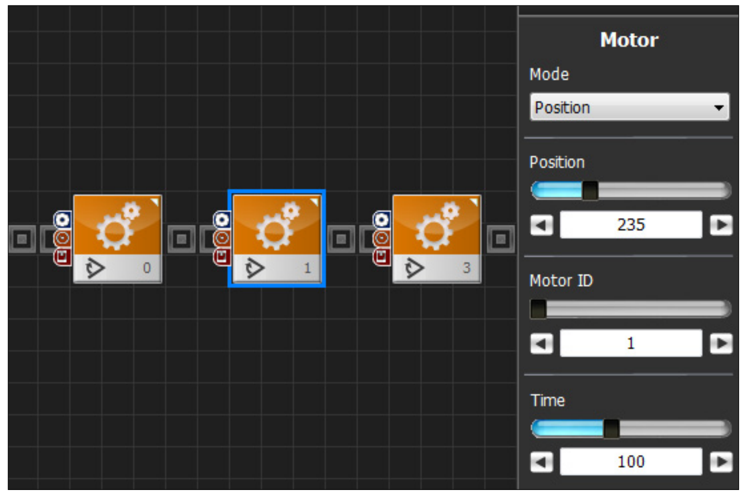


Motor
Mode: Position
Position: 235
Motor ID: 0
Time: 100

09 Moto ID 0 (Right Shoulder) Setup

When all servo motors are aligned to the center, humanoid robot will be standing with both arms stretched out to the side. In order to make applying motion easier, robot posture should be returned to the basic attention posture with both arms lowered to the side of the body.

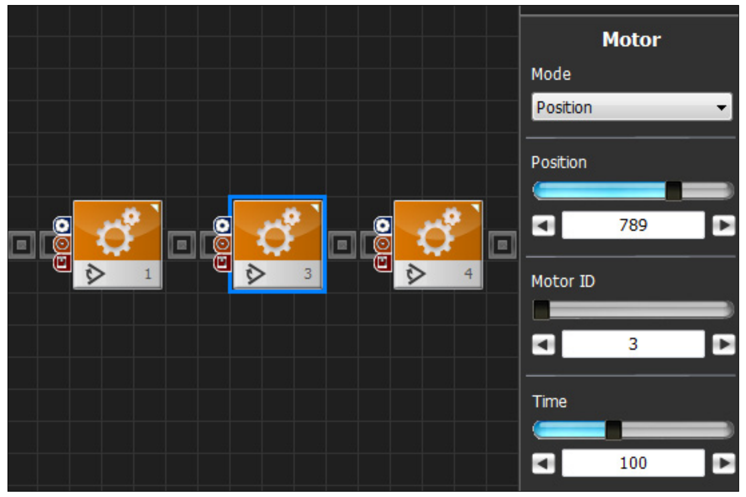
Select Motion > Motor and place it after the previous module.
Set Mode : Position, to control the angle.
Set Position : 235. Changing the motor position to 235 will allow the horizontally stretched arm to be lowered to vertical



Motor
Mode: Position
Position: 235
Motor ID: 1
Time: 100

10 Set Motor ID 1 (Upper Right Arm)

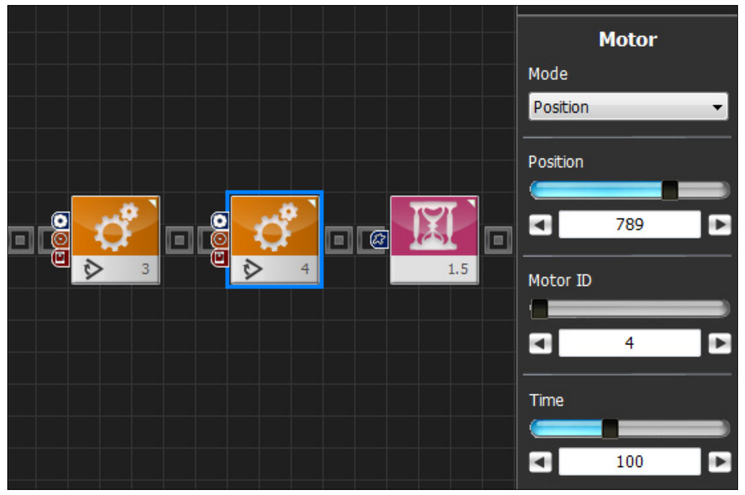
Select Motion > Motor module and place it after the previous module.
Set Mode : Position to control the motor position.
Set Position : 235. Changing the motor position to 235 will lower the horizontally stretched arm to the new vertical position.
Set Motor ID : 1, upper right arm motor has ID 1.
Set Time : 100, motor will move to the required position in approximately 1.12s.



Motor
Mode: Position
Position: 789
Motor ID: 3
Time: 100

11 Set Motor ID 3 (Left Shoulder)

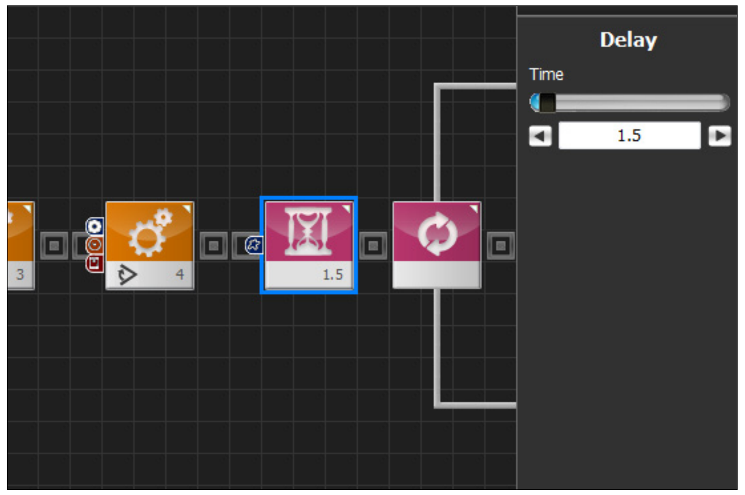
This section turns the left shoulder in order to lower the arm to the side of the body.
Select Motion > Motor module and place it after the previous module
Set Mode : Position to control the motor position.
Set Position : 789. Changing the motor position to 789 will allow the horizontally stretched arm to be lowered to vertical position.
Set Motor ID : 3 , left shoulder motor has ID 0.
Set Time : 100, motor will move to the required position in approximately 1.12s.



Motor
Mode: Position
Position: 789
Motor ID: 4
Time: 100

12 Set Motor ID 4 (Upper Left Arm)

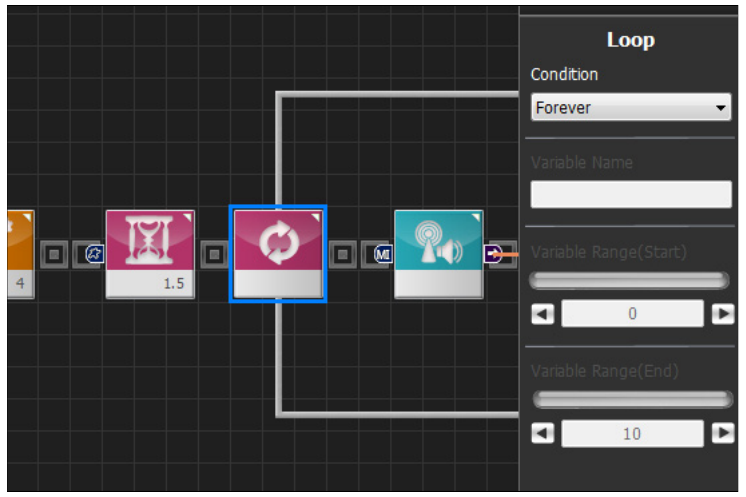
Select Motion > Motor module and place it after the previous module.
Set Mode : Position to control the motor position.
Set Position : 789. Changing the motor position to 789 will lower the horizontally stretched arm to the new vertical position.
Set Motor ID : 4, upper left arm motor has ID 4.
Set Time : 100, motor will move to the required position in approximately 1.12s.


Delay
Time
1.5

13 Delay

This section adds 1.5s of delay before starting the next motor operation. Delay allows the motor to move to the position requested by the previous module without any interruption. Individual motion command to the motors 0 1 3 4 were sent immediately after the command to send all motors (254) to position 512. From users perspective, it would seem like robot lowered both arms simultaneously to take attention posture.

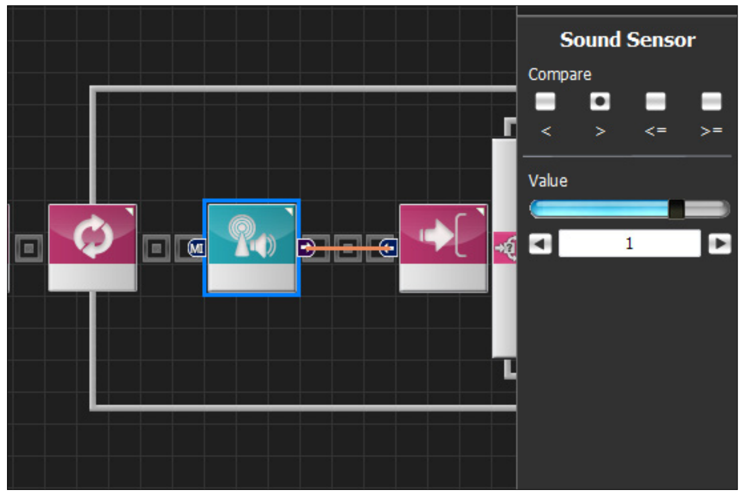
Select Flow > Delay module and place it after the previous module.
Set Time : 1.5s, delay 1.5s.


Loop
Condition
Forever
Variable Name
Variable Range(Start)
0
Variable Range(End)
10

14 Loop Statement

Insert loop statement to run the program in infinite loop.

Select Flow > Loop
Select Condition : Forever to create infinite loop statement without any condition.

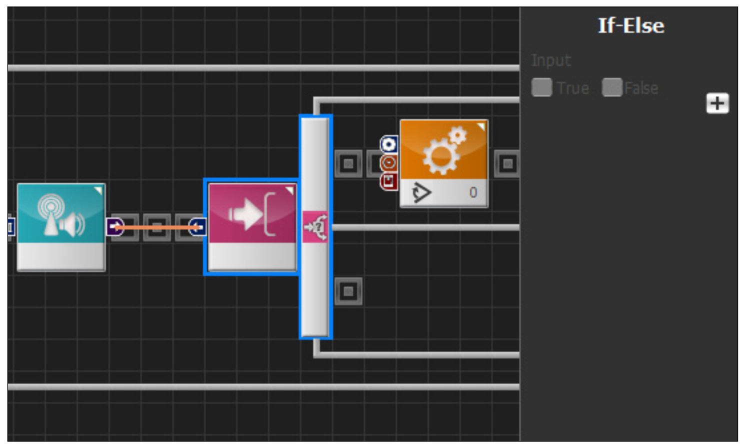

Sound Sensor
Compare
< > <= >=
Value
1

15 Add Sound Sensor

Add Sound sensor module. This module will output True/False after comparing the direction of the detected sound with the Value.

Select Sensor > Sound and place it within the loop module.

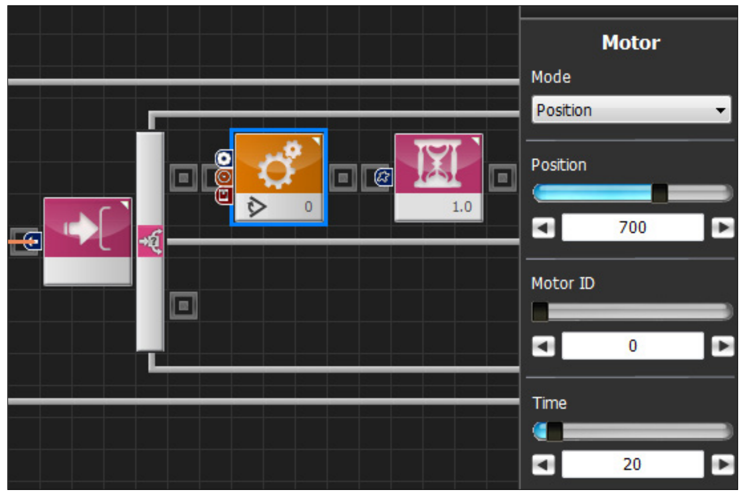
Select Compare : > . Output 'True' if Sensor value is greater than Value.
Set Value : 1. 1 means sound detected from the right side. Output 'True' if sensor value is greater than 1 and output 'False' if value is equal to or less than 1. In other words, value becomes 'True' if sound is from the right side.


If-Else
Input
True False

16 If-Else Statement

Create conditional statement to produce different motion depending on the output of the sound sensor.

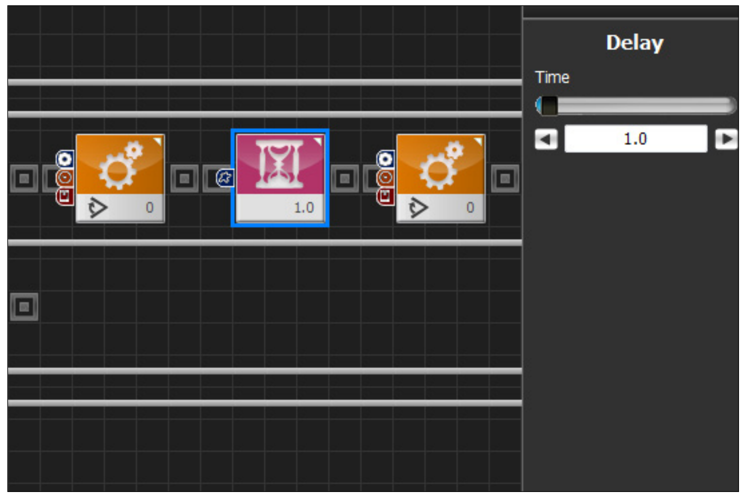
Select Flow > If-Else and place the module after the previous module.
Connect the output pin from the sound sensor in step 15 to the If-Else input pin.


Motor
Mode
Position
Position
700
Motor ID
0
Time
20

17 Motor ID 0 (Right Shoulder) Setup

Lift up the right arm if the sound sensor value is greater than 1 (True). In other words, lift right arm if sound was detected from the right side.

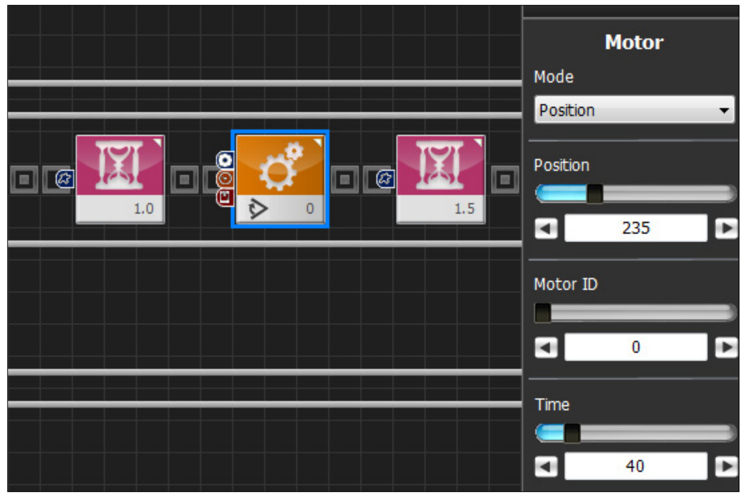
Select Motion > Motor and place it in the upper program line in the If-Else statement.
Set Mode : Position
Set Position : 700. Changing the motor position to 700 will lift the lowered arm straight forward.
Set Motor ID : 0, right shoulder motor has ID 0.
Set Time : 20


Delay
Time
1.0

18 Motion Delay

Wait for the arm to lift as it takes time for the arm to actually finish lifting after the lift command is given. Add 1s of delay to add slight pause after arm finishes lifting.

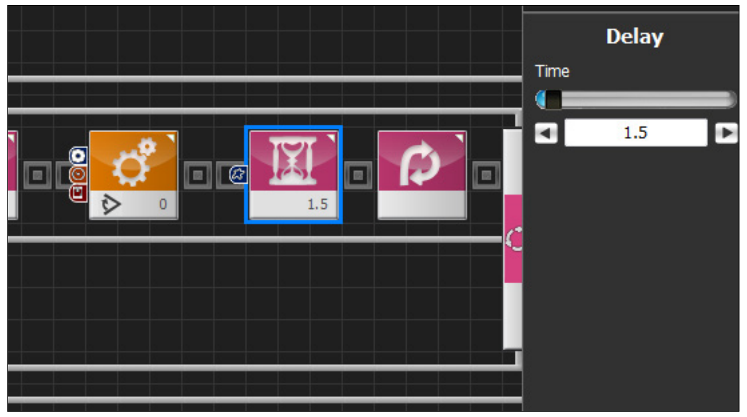
Select Flow > Delay and place it after the previous module.
Set Time : 1.0. Wait 1s.



19 Lower Motor ID 0 (Right Shoulder)

Lower right arm from the lifted position.

Set Mode : Position
Set Position : 235. Changing the motor position to 235 will lower the arm to the side of the body.
Set Motor ID : 0, right shoulder motor has ID 0.
Set Time : 40. Arm will be lowered at a slightly slower pace (0.448s) than when it was being raised.

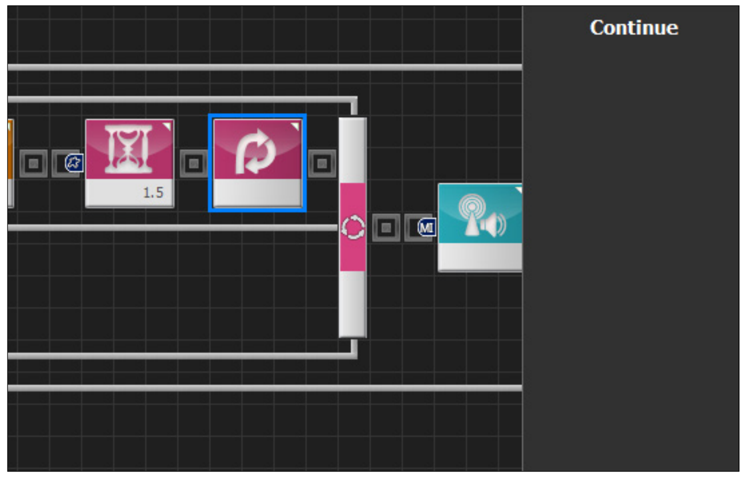


20 Motion Delay

It takes time for arm to lower and also for sound sensor value to initialize after the motion. Add delay of 1.5s to add more than 1s of delay after the motion ends.

Select Flow > Delay and place the module after the previous module.

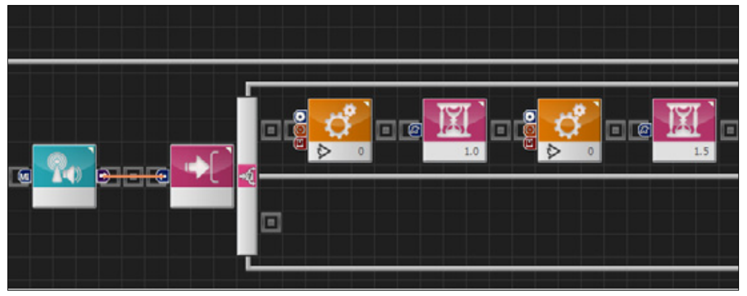
Set Time : 1.5. Delay 1.5s.



21 Place Continue

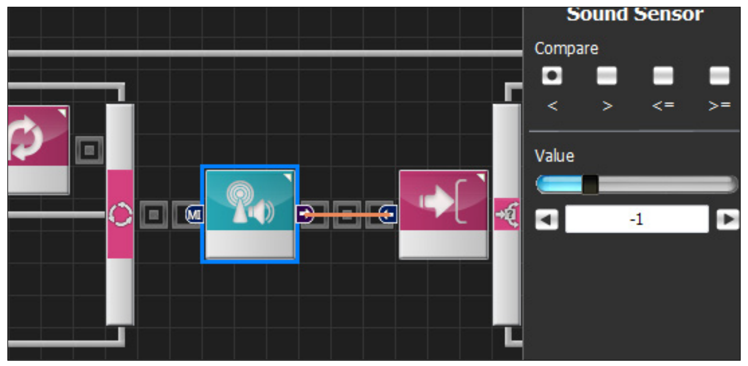
Add Continue to go back to the beginning of the loop.

Select Flow > Continue and place it after the previous module.



22 Summary

Conditional branch statement was added to block external stimulus while robot is in motion following sound input from the right side. This addition results in much greater accuracy by preventing sound input from surrounding noise or from the robot motors while robot is in motion.

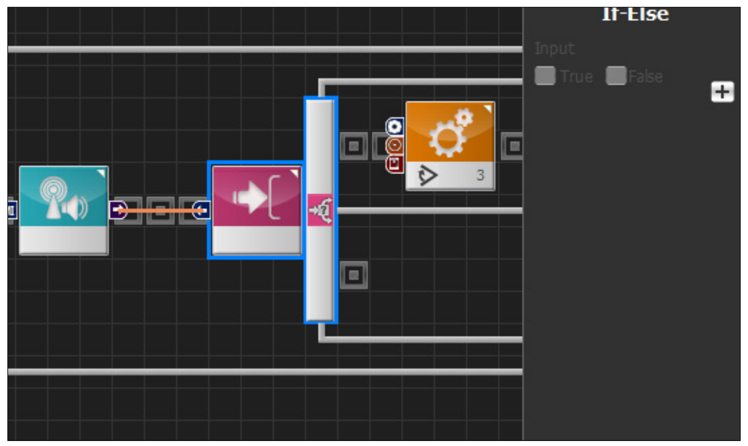


23 Add Sound Sensor

Add Sound sensor module. This module will output True/False after comparing the direction of the detected sound with the Value.

Select Sensor > Sound and place it within the loop module.

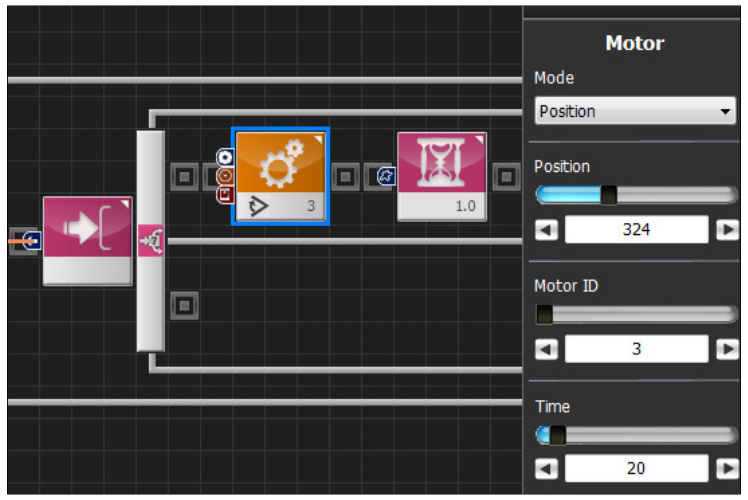
Select Compare : < . Output 'True' if Sensor value is less than Value.
Set Value : -1. -1 means sound detected from the left side. Output 'True' if sensor value is less than -1 and output 'False' if value is equal to or greater than -1. In other words, value becomes 'True' if sound is from the left side.



24 If-Else Statement

Create conditional statement to produce different motion depending on the output of the sound sensor.

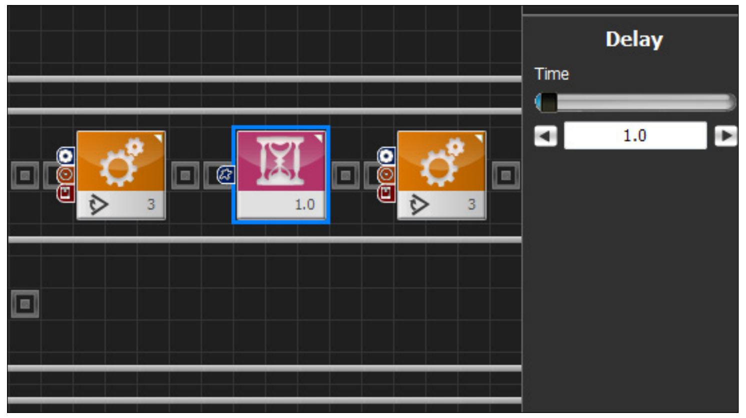
Select Flow > If-Else and place the module after the previous module.
Connect the output pin from the sound sensor in step 23 to the If-Else input pin.


Motor
Mode
Position
Position
324
Motor ID
3
Time
20

25 Motor ID 3 (Left Shoulder) Setup

Lift up the left arm if the sound sensor value is less than -1 (True). In other words, lift left arm if sound was detected from the left side.

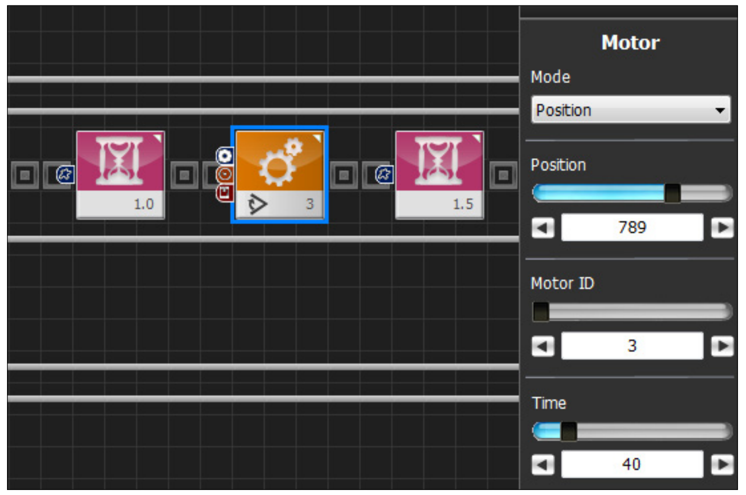
Select Motion > Motor and place it in the upper program line in the If-Else statement..
Set Mode : Position
Set Position : 324. Changing the motor position to 324 will lift the lowered arm straight forward.
Set Motor ID : 3, left shoulder motor has ID 3.
Set Time : 20 (0.224s)


Delay
Time
1.0

26 Motion Delay

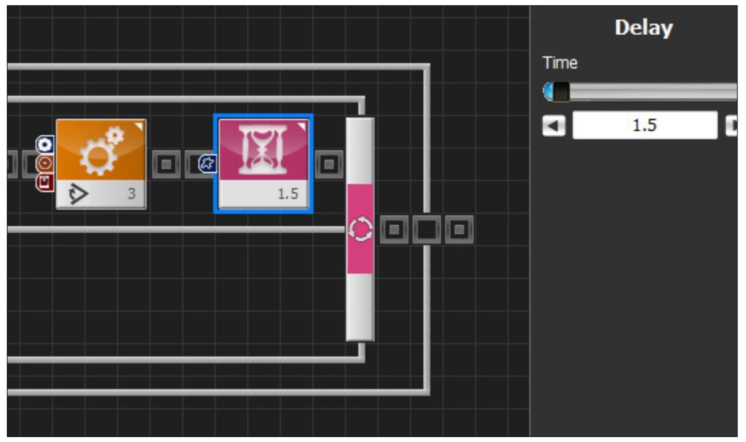
Wait for the arm to lift as it takes time for the arm to actually finish lifting after the lift command is given. Add 1s of delay to add slight pause after arm finishes lifting.

Select Flow > Delay and place it after the previous module.
Set Time : 1.0. Wait 1s.


Motor
Mode
Position
Position
789
Motor ID
3
Time
40

27 Motor ID 3 (Left Shoulder) Setup

Lower the left arm if no sound was detected or if the sound sensor value is equal to or greater than -1 (False). In other words, lower arm if no sound is detected from the left side.
Select Motion > Motor and place it in the lower program line in the If-Else statement.
Set Mode : Position
Set Position : 789. Changing the motor position to 789 will lower the arm to the side of the body.
Set Motor ID : 3 , left shoulder motor has ID 3.
Set Time : 40. Arm will be lowered at a slightly slower pace than when it was being raised.

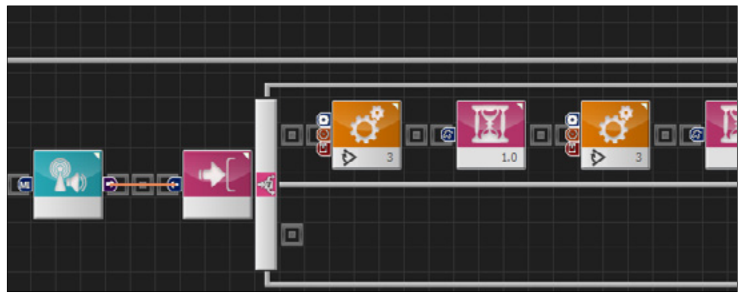

Delay
Time
1.5

28 Motion Delay

It takes time for arm to lower and also for sound sensor value to initialize after the motion. Add delay of 1.5s to add more than 1s of delay after the motion ends.

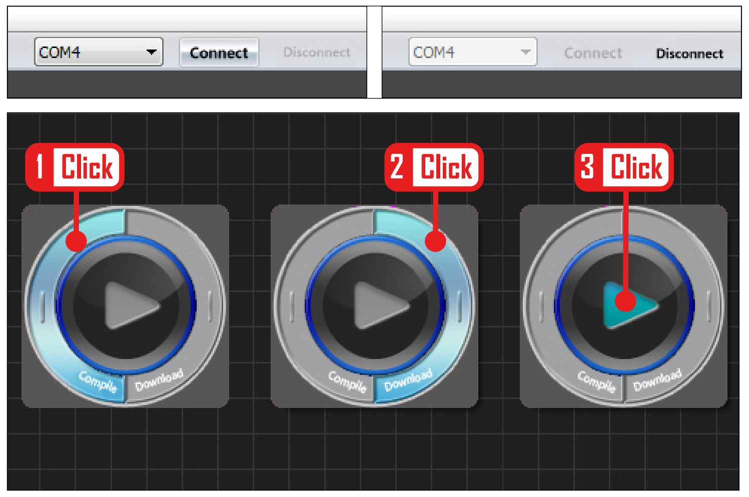
Select Flow > Delay and place the module after the previous module.

Set Time : 1.5. Delay 1.5s



29 Summary

Conditional branch statement was added to block external stimulus while robot is in motion following sound input from the left side. This addition results in much greater accuracy by preventing sound input from surrounding noise or from the robot motors while robot is in motion.

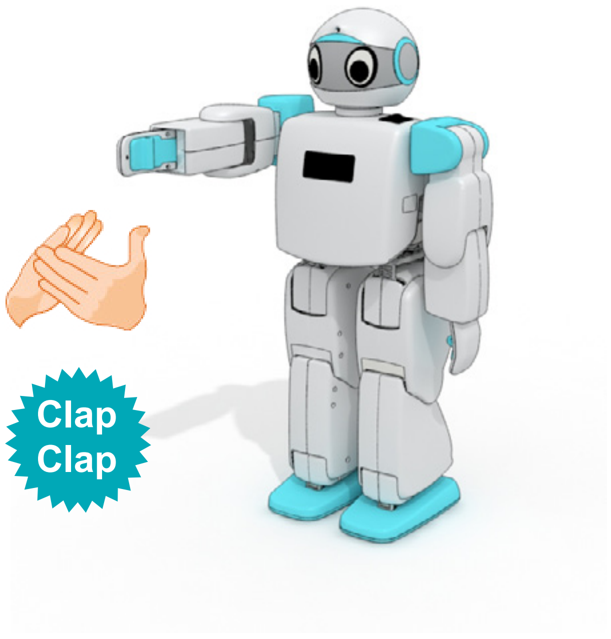


30 Download

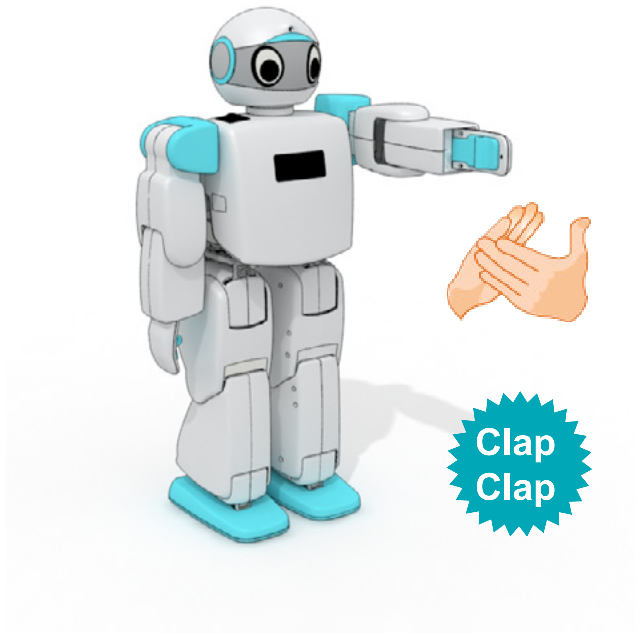
After programming is completed.
Compile -> Download -> Run.

Select the appropriate COMPORT or USB-to-Serial number located on the PC and press Connect to open the serial port. Click 'Compile' and then click 'Download' on the right to download the program to the robot if no compile error is found. Click "Run" button (arrow in the middle) after completing the download.

1



2



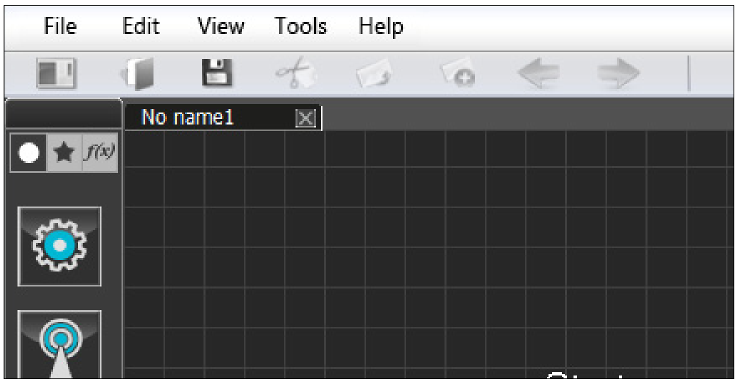
31 Robot Motion

Robot will lift the left arm with left side clap and right arm with the right side clap.

08-7 Programming Individual Module Digtal Distanc

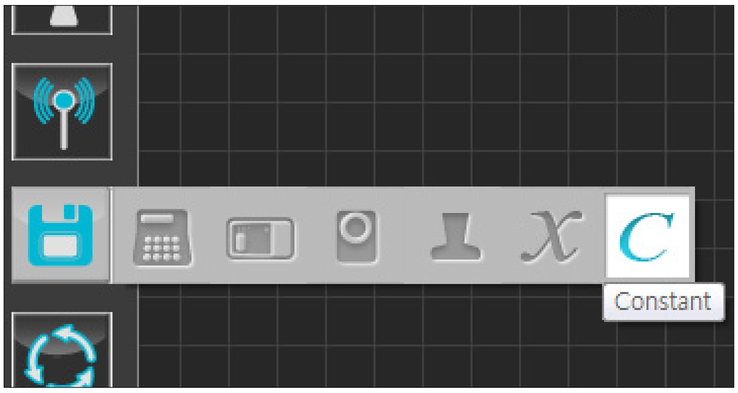
Example Description

Analog distance sensors are capable of measuring the distance within the valid range. Whereas digital sensors are used to detect whether an obstacle is near or far relative to the specified standard distance. Digital sensors are often attached to wheeled robots with the sensors facing the floor to detect sudden drop on the floor and also to humanoid robot knees to detect obstacles. In this example program, robot will walk backwards, turn direction, and then start walking forward again when it detects an obstacle within certain distance. This example requires digital sensor to be attached to the ADC port #1 (left side) with the sensor facing forward direction.



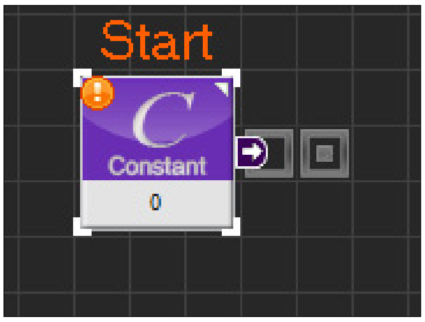
01 Create New File

Click on the left most icon in the tools to create new dts file. Select HOVIS Lite/ Eco as the Robot.



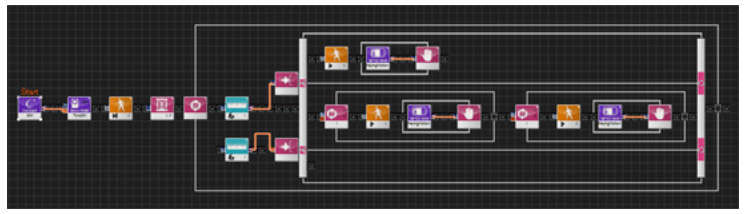
02 Select Module

Prior to placing the module, click to select the module from the module bar located on the left side.
Click Data > Constant module



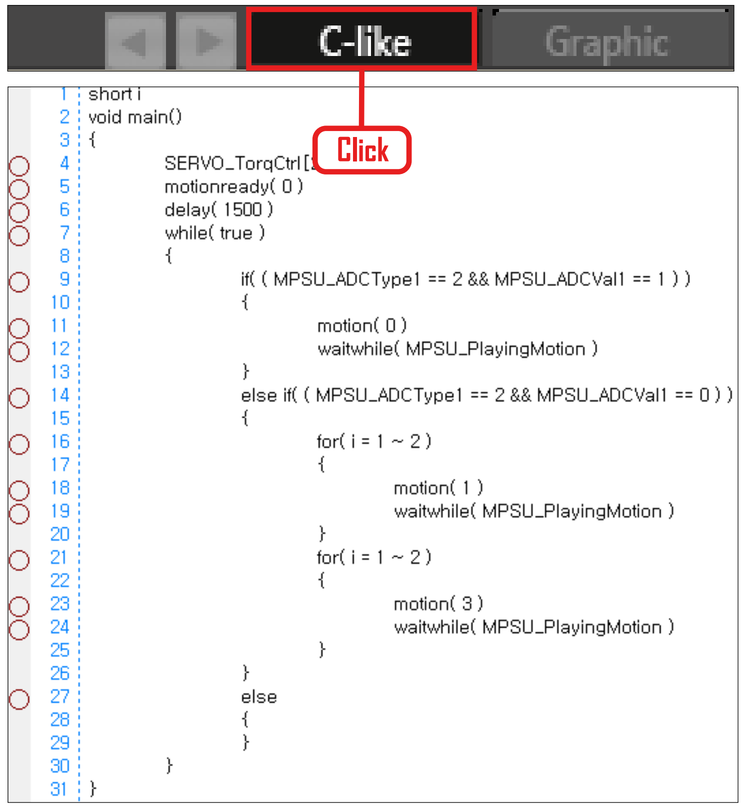
03 Place Module

Drag and dock the module to the Start Point to activate the module. Active module will turn to color.



04 Entire Program

View of the entire program.

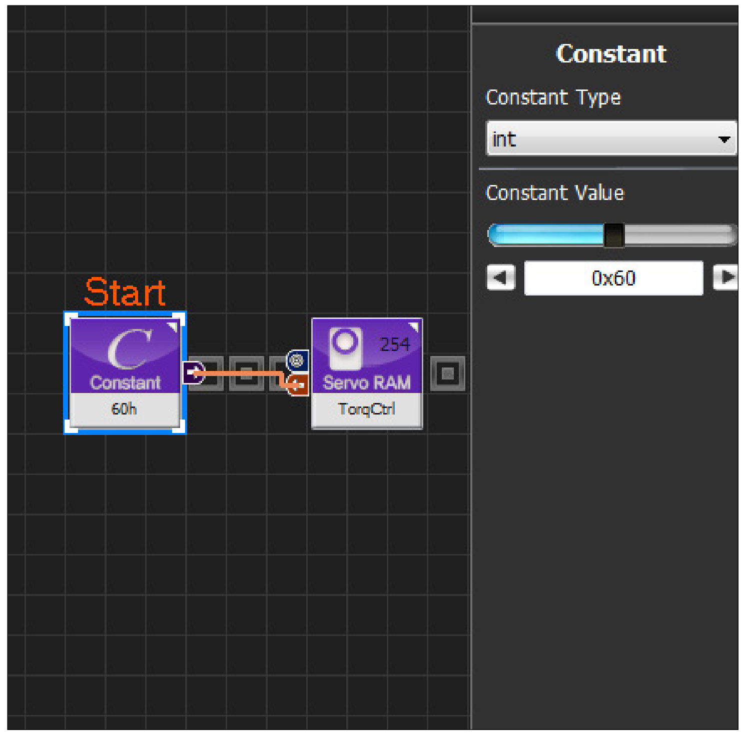


```
short i
void main()
{
        SERVO_TorqCtrl[
        motionready( 0 )
        delay( 1500 )
        while( true )
        {
                if( ( MPSU_ADCType1 == 2 && MPSU_ADCVal1 == 1 ) )
                {
                        motion( 0 )
                        waitwhile( MPSU_PlayingMotion )
                }
                else if( ( MPSU_ADCType1 == 2 && MPSU_ADCVal1 == 0 ) )
                {
                        for( i = 1 ~ 2 )
                        {
                                motion( 1 )
                                waitwhile( MPSU_PlayingMotion )
                        }
                        for( i = 1 ~ 2 )
                        {
                                motion( 3 )
                                waitwhile( MPSU_PlayingMotion )
                        }
                }
                else
                {
                }
        }
}
```

05 Viewing C-Like

Click “C-Like” tab near the top right to open the program source code window as shown in the left.

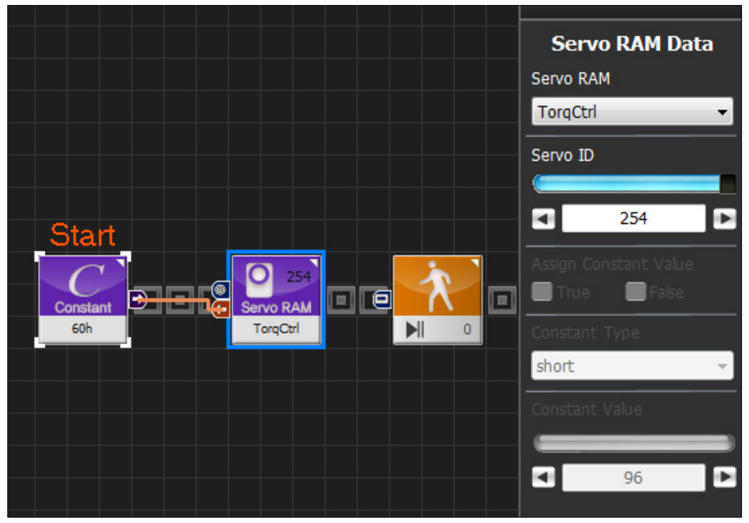
Since codes are very similar to the C language structure, studying the codes will help the user become familiar with the C language as well. Clicking on the module results in cursor jumping to the matching source code line in the C-Like window. This feature enables the user to easily see the codes generated by the module.



Constant
Constant Type
int
Constant Value
0x60

06 Set Constant

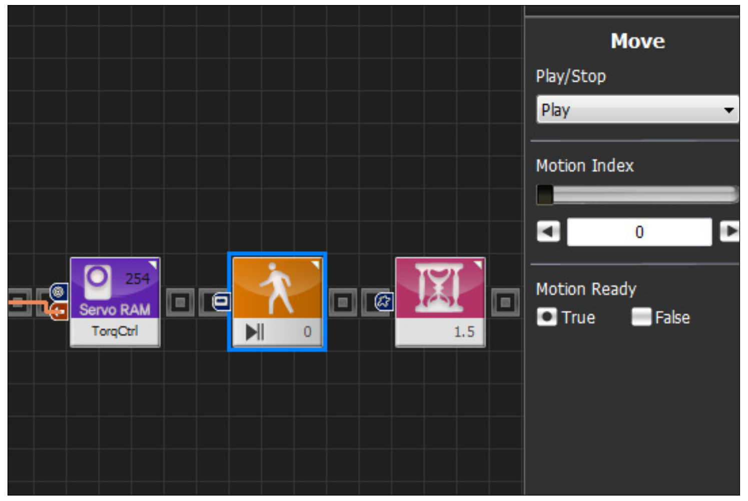
This section allows the servo motor to operate on its own. From the Constant module properties, click on the Constant Value and change the value to 0x60. Constant with Hex number of 0x60 shows the torque enabled state of the servo motor. This value is sent to the input pin of the next module through the output pin.



Servo RAM Data
Servo RAM
TorqCtrl
Servo ID
254
Assign Constant Value
True False
Constant Type
short
Constant Value
96

07 Apply to All servos

This section applies constant value 0x60 received from the previous section to all servo motors.
Select Data > Servo RAM and place it after the Constant module.
Servo RAM : Select TorqCtrl
Servo ID: Set to 254, ID 254 means set-up will be applied to all connected servos.
Use the Connector to connect the output pin of the Constant module to the second input pin of the Servo RAM module.



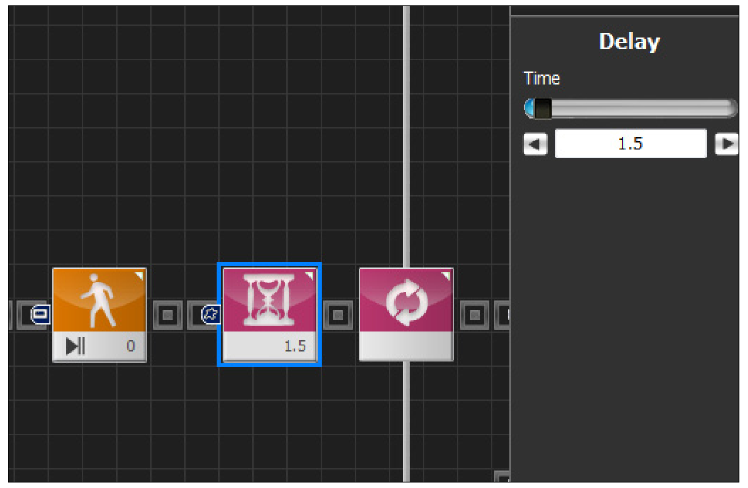
Move
Play/Stop
Play
Motion Index
0
Motion Ready
True False

08 Motion Ready

When motion is first loaded, robot may make a sudden movement. If the difference between the current position and the start of the motion is very different, it may cause stress to the motor or pose danger to the user. To prevent such an occurrence, motion is run in ‘Motion Ready’ mode to provide time for motion to start.

Select Motion > Move module and place it after the previous module.
Play/Stop: Select Play.
Select Motion Index :0, load motion number 0 which is a forward walk motion.

Select Motion Ready: True. When set to True, robot will slowly move to starting position of the motion.

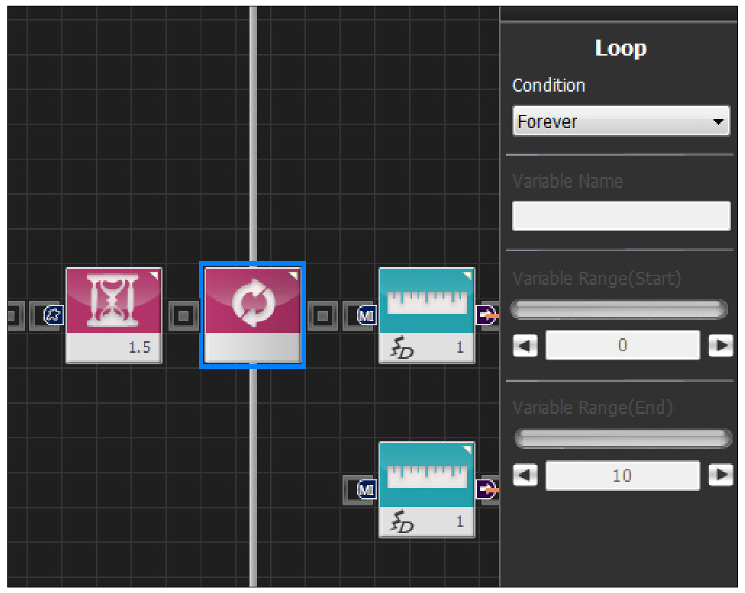


Delay
Time
1.5

09 Delay

Set Delay value to 1.5s to prevent robot motion from starting before the Motion Ready ends.

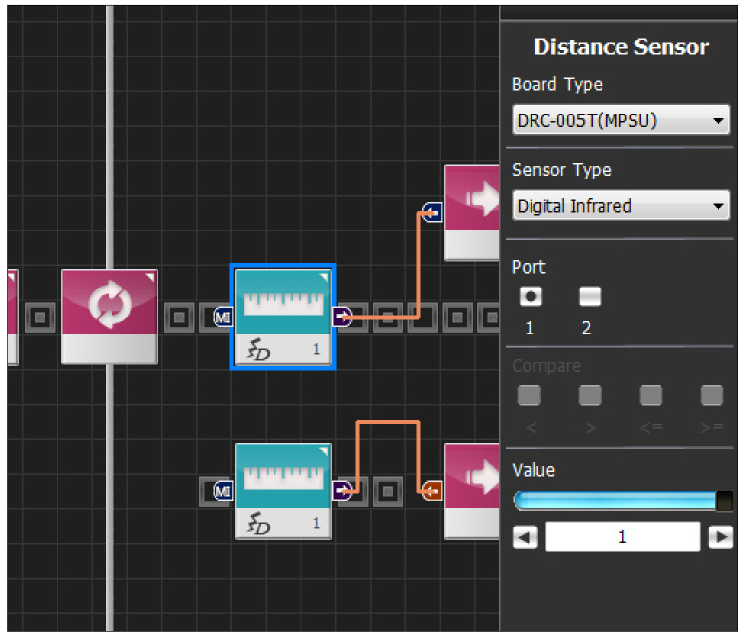
Select Flow > Delay module and place it after the previous module.
Set Time: 1.5. Delay 1.5s.



Loop
Condition: Forever
Variable Name
Variable Range(Start): 0
Variable Range(End): 10

10 Loop

Place loop module and set it to infinite loop mode to continuously repeat the motion. Content of the loop module will be repeated continuously.

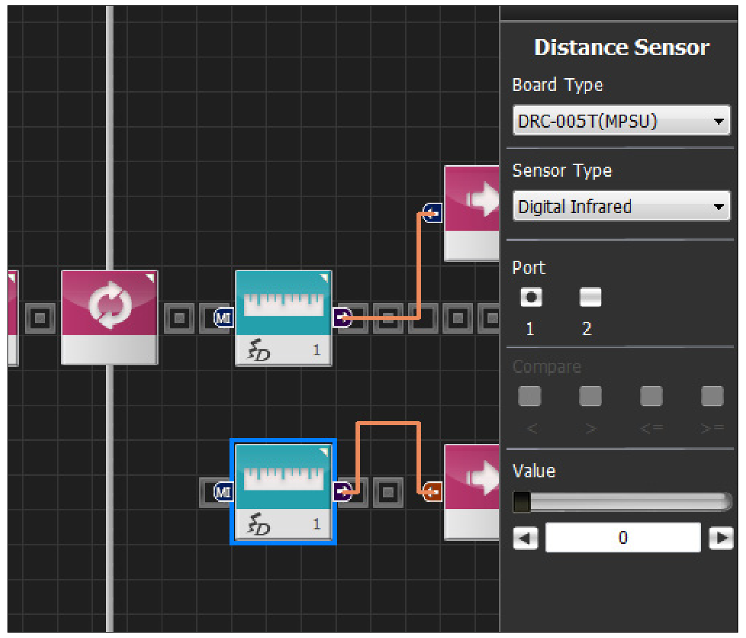


Distance Sensor
Board Type: DRC-005T(MPSU)
Sensor Type: Digital Infrared
Port: 1 2
Compare: < > <= >=
Value: 1

11 Add Distance Sensor

Add Digital distance sensor module within the loop statement. Digital sensor module has value of 1 if the detected distance value is greater than 10cm and 0 if closer than 10cm. This module is connected to the digital sensor module of the selected port and outputs 'True' if value is equal to Value. Output is 'False' if sensor is not connected.

Select Sensor > Distance and place the module inside the Loop module.
Select Board Type : DRC-005T(MPSU), select sensor connected to the controller DRC.
Select Sensor Type : Digital Infrared (Digital infrared distance sensor)
Select Port : 1, There are two ports, left port is port 1.
Set Value : 1

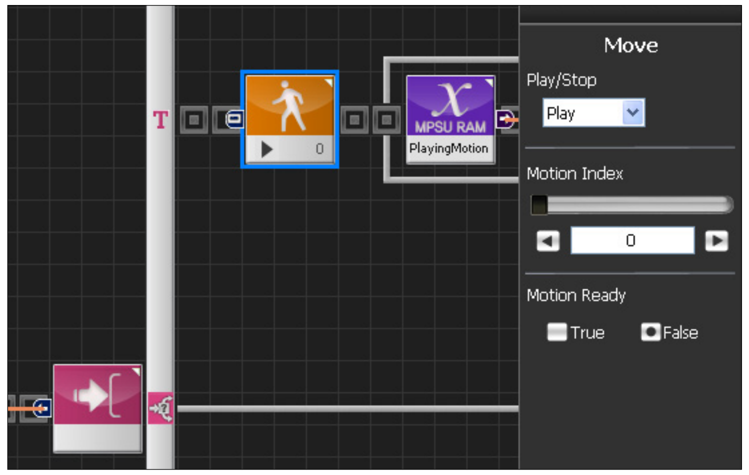


Distance Sensor
Board Type: DRC-005T(MPSU)
Sensor Type: Digital Infrared
Port: 1 2
Compare: < > <= >=
Value: 0

12 Add Distance Sensor

Add one more digital distance sensor module and place it below the sensor module added in step 11. Unlike the previous module, Value for this module is 0.

Select Sensor > Distance and place the module below the module in step 11.
Select Board Type : DRC-005T(MPSU),
Select Sensor Type : Digital Infrared (Digital infrared distance sensor)
Select Port : 1
Set Value : 0

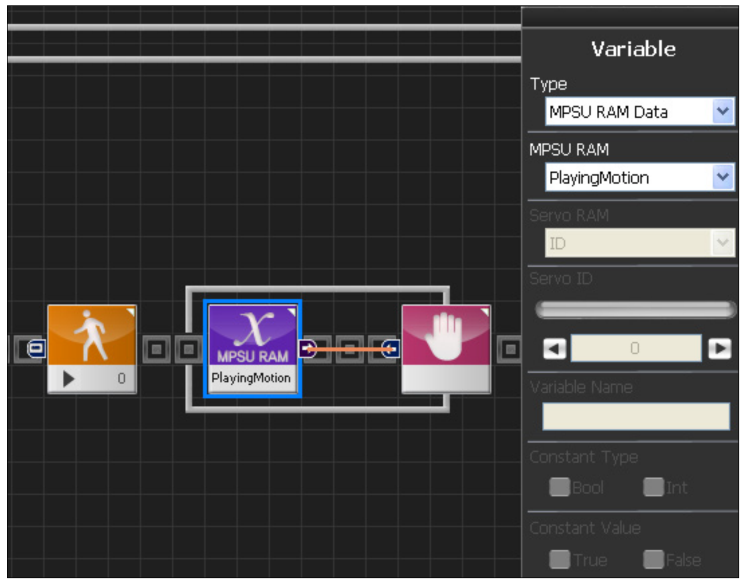


Move
Play/Stop: Play
Motion Index: 0
Motion Ready: True False

13 Create If-Else Statement

Create If-Else statement and connect it to the previously made sensor module If-Else statement.

Select Flow > If-Else and place the module within the Loop module.
Add one more conditional statement by clicking + shaped icon.
Connect the distance sensor module output pin from step 11 to the first input pin and output pin from step 12 to the second input pin.



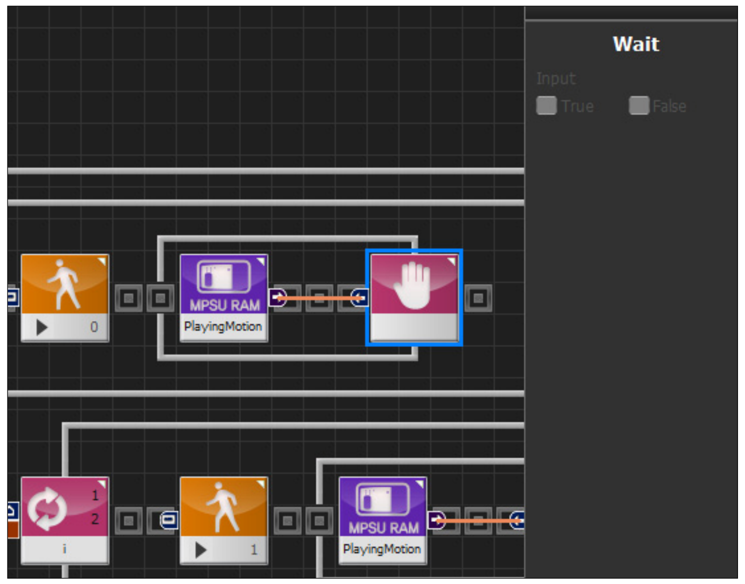
Variable
Type: MPSU RAM Data
MPSU RAM: PlayingMotion
Servo RAM: ID
Servo ID: 0
Variable Name
Constant Type: Bool Int
Constant Value: True False

14 Run Forward Motion

Add Move module for running motion #0 (forward) to the first program line of the If-Else module. Robot will move forward if the distance value is greater than 10cm (sensor value 1).

Select Motion > Move and place the module on the first program line of the If-Else module.

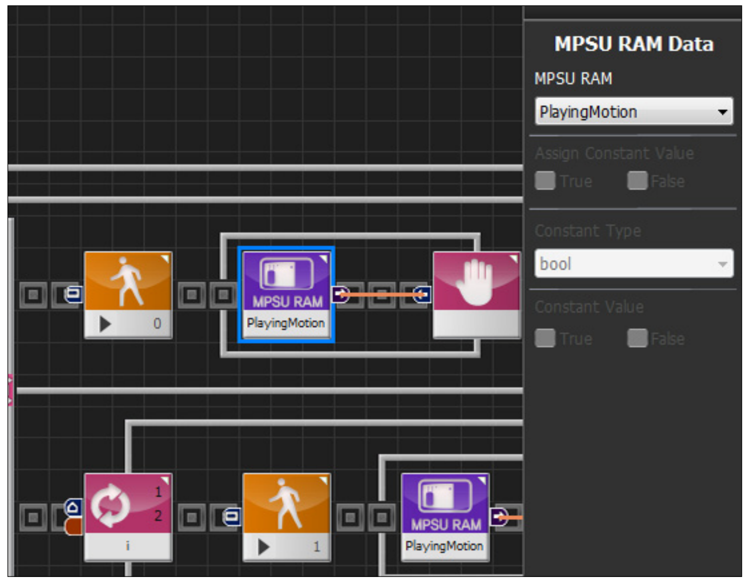
Select Play/Stop : Play
Select Motion Index : 0. Play motion # 0 (forward motion).
Select Motion Ready : False.



15 Add Wait.

Add Wait module after the Move module to wait until the motion ends.

Select Flow > Wait and place the module after the Move module in step 14.


MPSU RAM Data

MPSU RAM: PlayingMotion

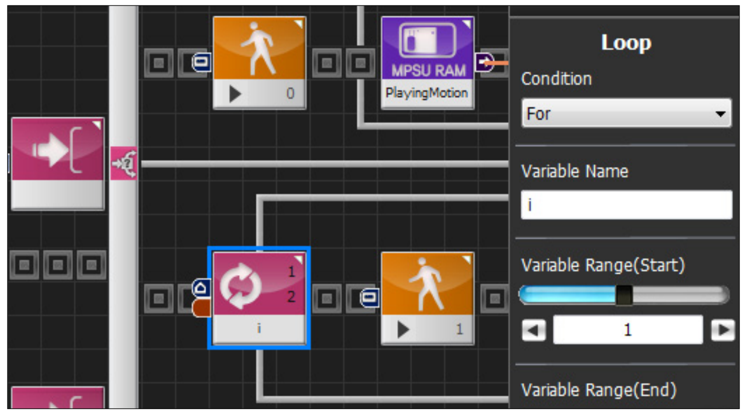
Constant Type: bool

16 Add PlayingMotion Module

If loop contains only single move module, move module in the loop will issue a motion command even while the previous motion is still running. This results in difference between the number of motion commands given and actual number of motions played. In order to correct this discrepancy, delay should be introduced to delay the loop from going back to the beginning until the current motion ends. Data > MPSU RAM contains “PlayingMotion”. This register is a variable which has value of 1 if motion is in progress and value of 0 if motion is not active. Connect ‘PlayMotion’ to Wait module input pin to pause the program at the Wait module while the motion is playing.

Select Data > MPSU RAM and place the module inside the Wait module.

Select MPSU RAM : Palyingmotion. Connect the output pin of the module to the input pin of the Wait module.


Loop

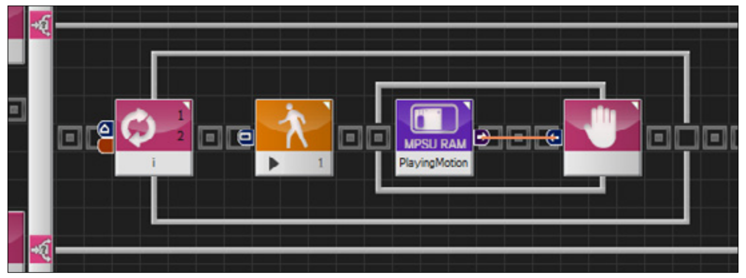
Condition: For

Variable Name: i

Variable Range(Start): 1

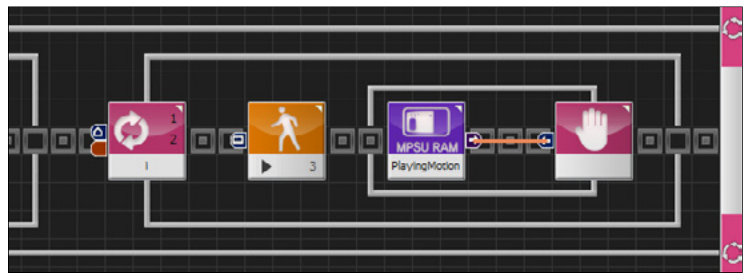
17 Create For statement

Loop module can be used to repeat the content of the loop for specified number of times as well as being used to repeat the loop infinitely. Select For statement after adding the loop module and then select variable name and range. Variable value will increase by 1 within the range each time loop is repeated.



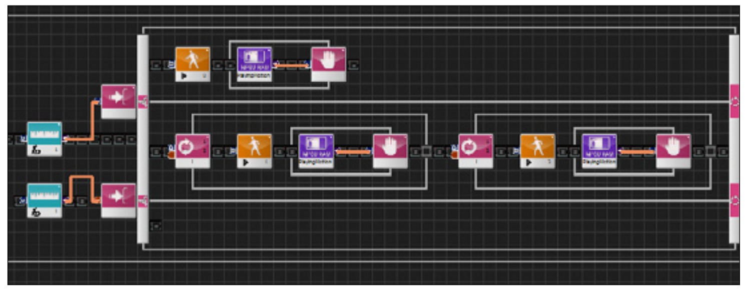
18 Ready Backward Motion

Follow the steps 14 ~ 16 and add motion #1 (backward) into the Loop module to create backward motion with delay. Backward motion #2 will be run when program encounters this For statement.



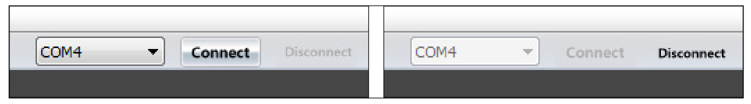
19 Repeat Right Turn Twice

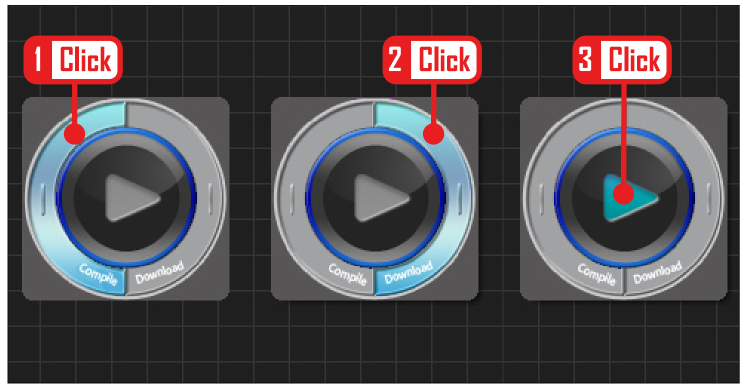
Use the same method to add motion #3 (right turn) after the for statement in steps 17~18 and repeat the right turn twice.



20 Summary

This is the shape of the entire If-Else statement we have created. If distance sensor value is 1 (greater than 10cm), forward motion will be run. If distance sensor value is 0 (less than 10cm), backward motion will be run twice and then the right turn motion twice to avoid the obstacle. Program will go back to the beginning of the Loop statement after IF-Else statement ends and enter the If-Else statement again depending on the value of the distance sensor.

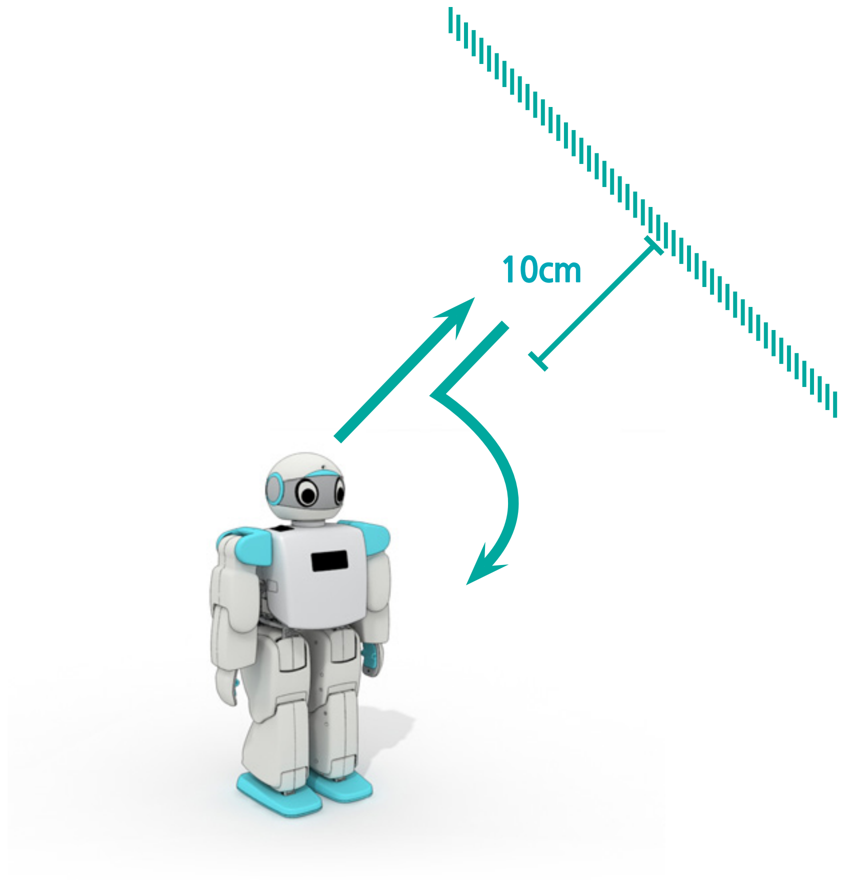




21 Download

After programming is completed.
Compile -> Download -> Run.

Select the appropriate COMPORT or USB-to-Serial number located on the PC and press Connect to open the serial port. Click ‘Compile’ and then click ‘Download’ on the right to download the program to the robot if no compile error is found. Click “Run” button (arrow in the middle) after completing the download.



22 Robot Motion

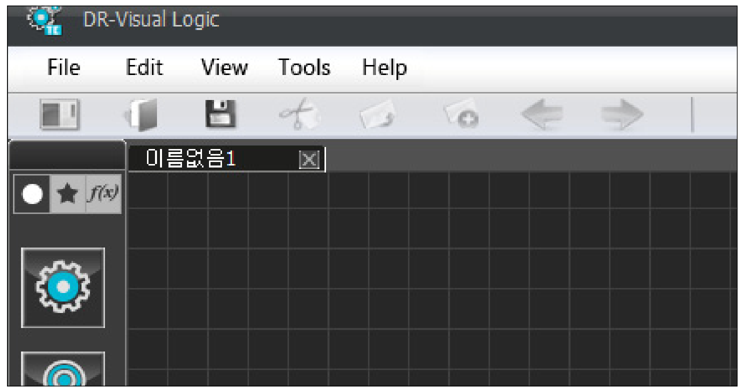
When detects a wall within 10cm, it will walk backwards, change direction to the right and start walking forward again.

Programming Individual Module

Analog Distance 08-8

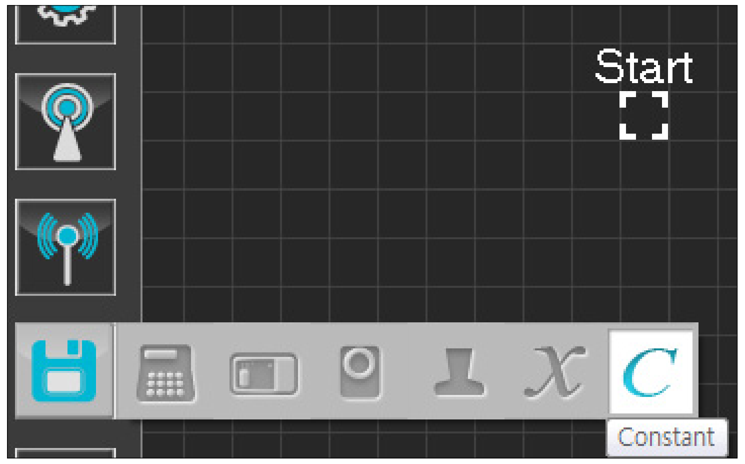
Example Description

Program will use the analog sensor to avoid an obstacle by making a left turn. There are two types of distance sensors, analog and digital. Whereas digital sensors can only detect whether an obstacle is within or outside of specific standard distance (10cm), analog distance sensors are able to measure actual the distance (6-40cm). In order run this example program, PSD sensor has to be connected to the ADC port #1 (left port) with the sensor facing forward direction.



01 Create New File

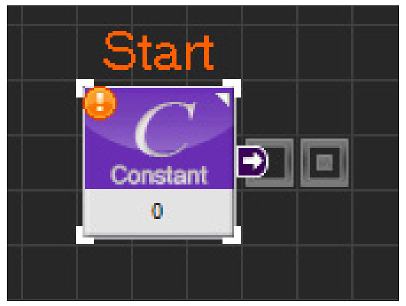
Click on the left most icon in the tools to create new dts file. Select HOVIS Lite/ Eco as the Robot.



02 Select Module

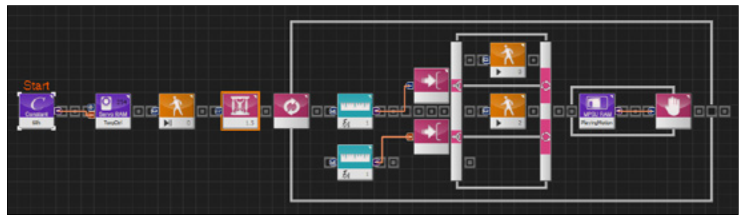
Prior to placing the module, click to select the module from the module bar located on the left side.

Click Data > Constant module



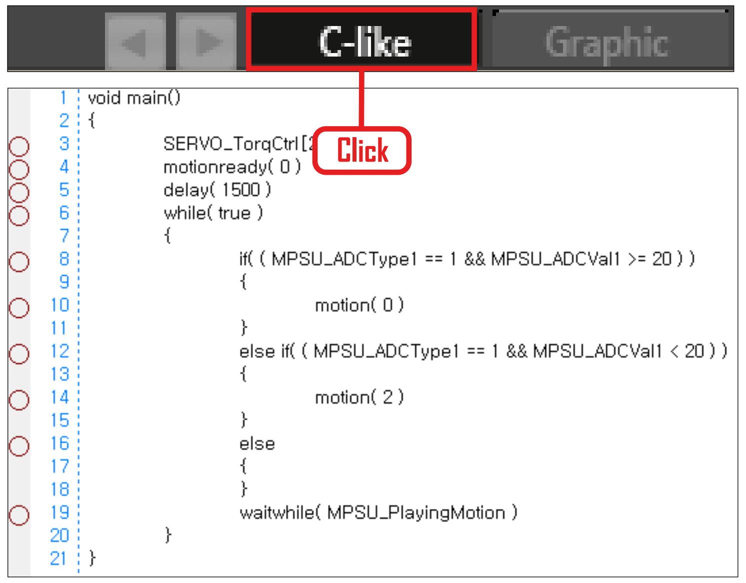
03 Place Module

Drag and dock the module to the Start Point to activate the module. Active module will turn to color.



04 Entire Program

View of the entire program

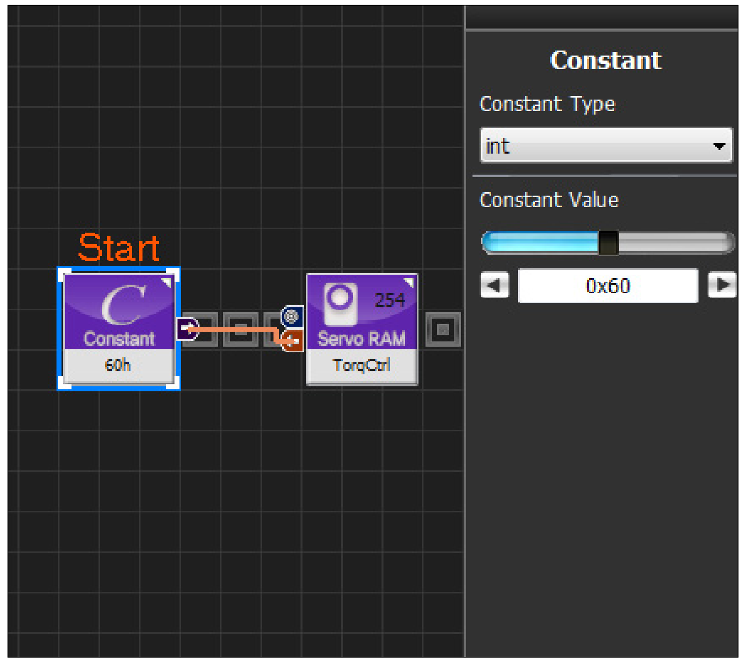


```
void main()
{
        SERVO_TorqCtrl[2
        motionready( 0 )
        delay( 1500 )
        while( true )
        {
                if( ( MPSU_ADCType1 == 1 && MPSU_ADCVal1 >= 20 ) )
                {
                        motion( 0 )
                }
                else if( ( MPSU_ADCType1 == 1 && MPSU_ADCVal1 < 20 ) )
                {
                        motion( 2 )
                }
                else
                {
                }
                waitwhile( MPSU_PlayingMotion )
        }
}
```

05 Viewing C-Like

Click "C-Like" tab near the top right to open the program source code window as shown in the left.

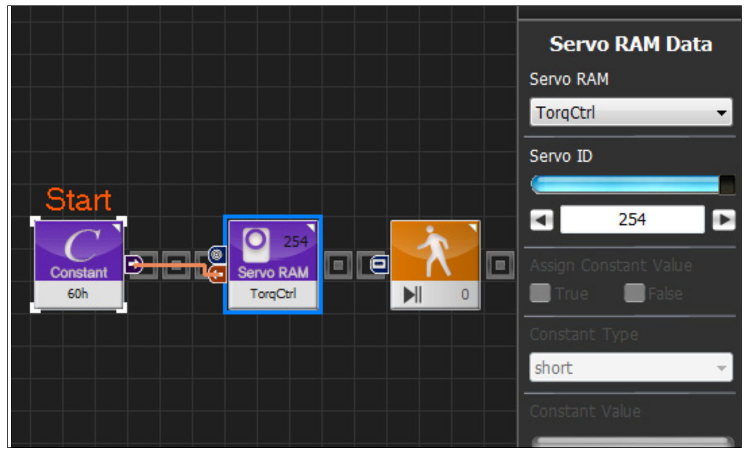
Since codes are very similar to the C language structure, studying the codes will help the user become familiar with the C language as well. Clicking on the module results in cursor jumping to the matching source code line in the C-Like window. This feature enables the user to easily see the codes generated by the module.



Constant
Constant Type: int
Constant Value: 0x60

06 Set Constant

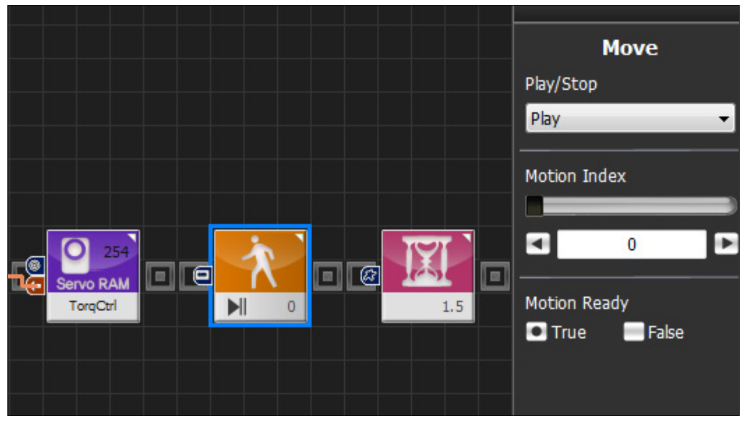
This section allows the servo motor to operate on its own. From the Constant module properties, click on the Constant Value and change the value to 0x60. Constant with Hex number of 0x60 shows the torque enabled state of the servo motor. This value is sent to the input pin of the next module through the output pin.



Servo RAM Data
Servo RAM: TorqCtrl
Servo ID: 254
Assign Constant Value: True False
Constant Type: short
Constant Value

07 Apply to All servos

This section applies constant value 0x60 received from the previous section to all servo motors.
Select Data > Servo RAM and place it after the Constant module.
Servo RAM : Select TorqCtrl
Servo ID : Set to 254, ID 254 means setup will be applied to all connected servos.
Use the Connector to connect the output pin of the Constant module to the second input pin of the Servo RAM module.



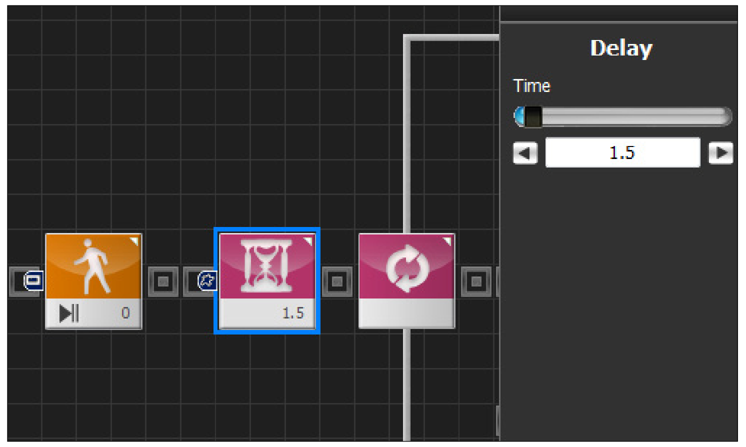
Move
Play/Stop: Play
Motion Index: 0
Motion Ready: True False

08 Motion Ready

When motion is first loaded, robot may make a sudden movement. If the difference between the current position and the start of the motion is very different, it may cause stress to the motor or pose danger to the user. To prevent such an occurrence, motion is run in 'Motion Ready' mode to provide time for motion to start.

Select Motion > Move module and place it after the previous module.
Play/Stop : Select Play.
Select Motion Index : 0, load motion number 0 which is a forward walk motion.

Select Motion Ready : True. When set to True, robot will slowly move to starting position of the motion.

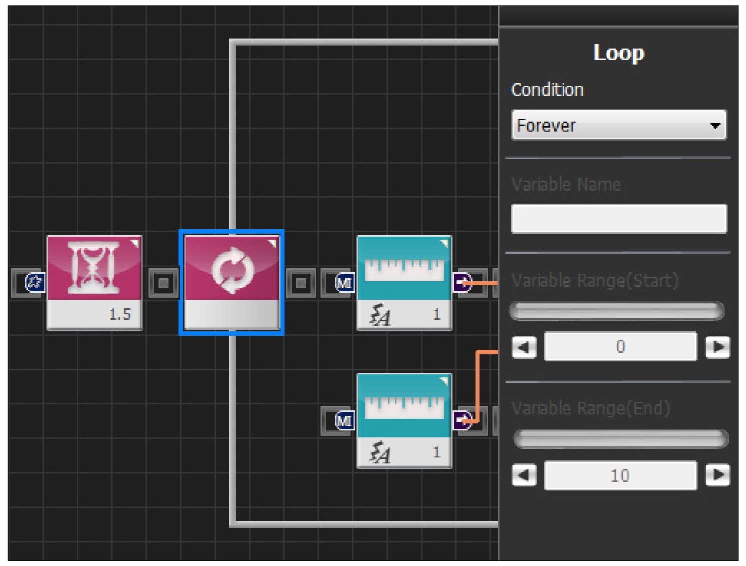


Delay
Time: 1.5

09 Delay

Set Delay value to 1.5s to prevent robot motion from starting before the Motion Ready ends.

Select Flow > Delay module and place it after the previous module.
Set Time : 1.5. Delay 1.5s.

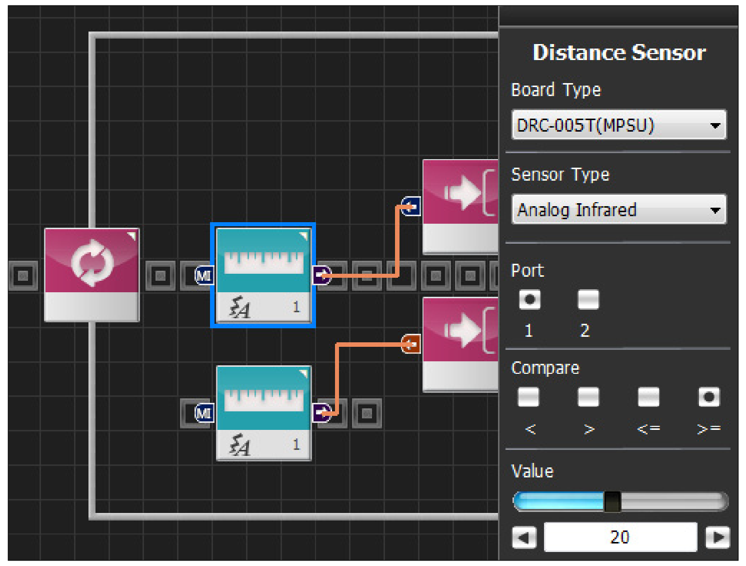


Loop
Condition: Forever
Variable Name
Variable Range(Start): 0
Variable Range(End): 10

10 Loop

Place loop module and set it to infinite loop mode to continuously repeat the motion. Content of the loop module will be repeated continuously.

Select Flow > Loop and place it after the previous module.
Set Condition : Forever, for infinite loop.

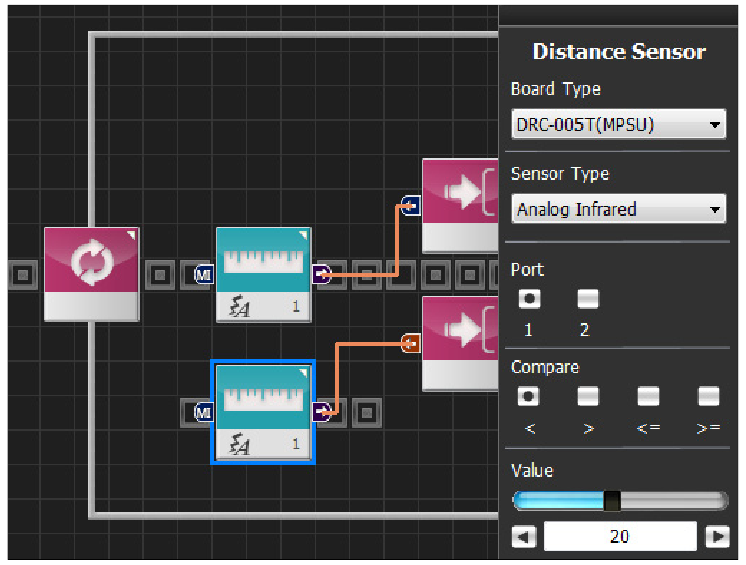


Distance Sensor
Board Type: DRC-005T(MPSU)
Sensor Type: Analog Infrared
Port: 1 2
Compare: < > <= >=
Value: 20

11 Add Distance Sensor

Add Analog distance sensor module within the loop statement. Analog sensor module outputs detected distance value in cm. This module is connected to the analog sensor module of the selected port and outputs True/False after using 'Compare' to compare the value with 'Value'.Output is 'False' if sensor is not connected.

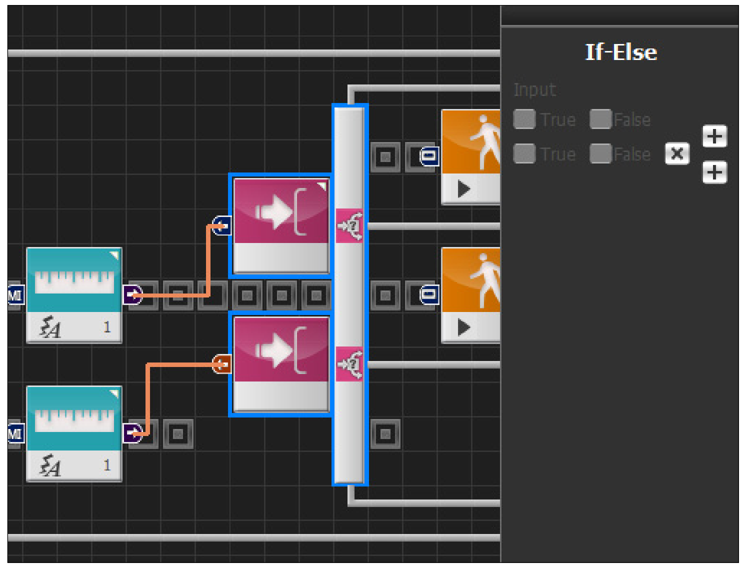
Select Sensor > Distance and place the module inside the Loop module.
Select Board Type : DRC-005T(MPSU), select sensor connected to the controller DRC.
Select Sensor Type : Analog Infrared (Analog infrared distance sensor)
Select Port : 1, There are two ports, left port is port 1.
Set Compare : >=. Output True when sensor value is greater than or equal to 'Value'.
Set Value : 20



12 Add Distance Sensor

Add one more Analog distance sensor module and place it below the sensor module added in step 11. Unlike the previous module, Compare will be set to <.

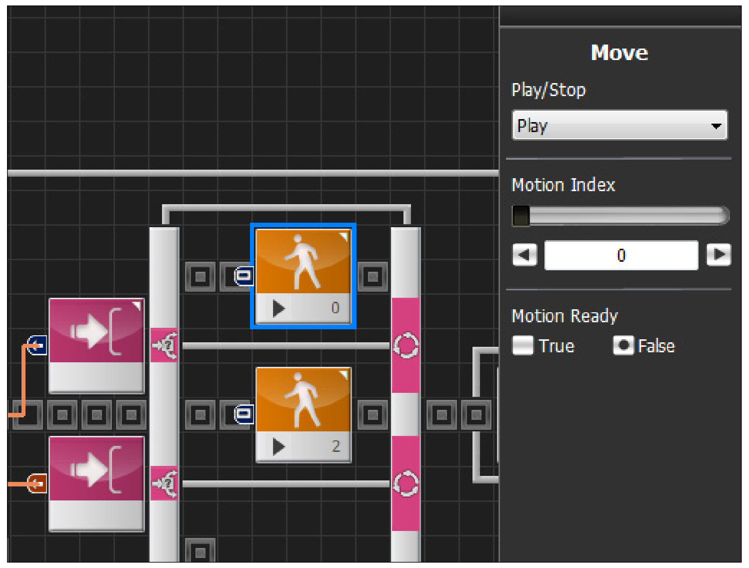
Select Sensor > Distance and place the module below the module in step 11.
Select Board Type : DRC-005T(MPSU),
Select Sensor Type : Analog Infrared (Analog infrared distance sensor)
Select Port : 1
Set Compare : <. Output True when sensor value is less than 'Value'.
Set Value : 20



13 Create If-Else Statement

Create If-Else statement and connect it to the previously made sensor module.

Select Flow > If-Else and place the module within the Loop module.
Add one more conditional statement by clicking + shaped icon.
Connect the distance sensor module output pin from step 11 to the first input pin and output pin from step 12 to the second input pin.

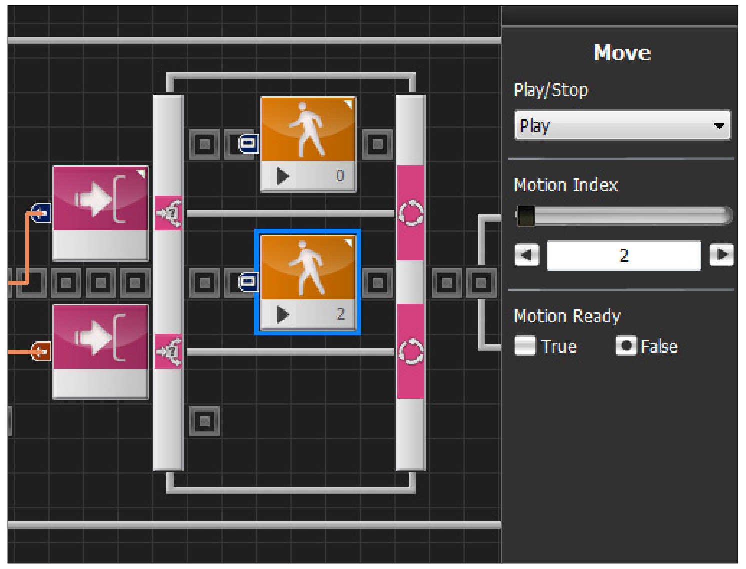


14 Run Forward Motion

Add Move module for running motion #0 (forward) to the first program line of the If-Else module. Robot will move forward if the distance value is greater than 20cm.

Select Motion > Move and place the module on the first program line of the If-Else module.

Select Play/Stop : Play
Select Motion Index : 0. Play motion # 0 (forward motion).
Select Motion Ready : False.

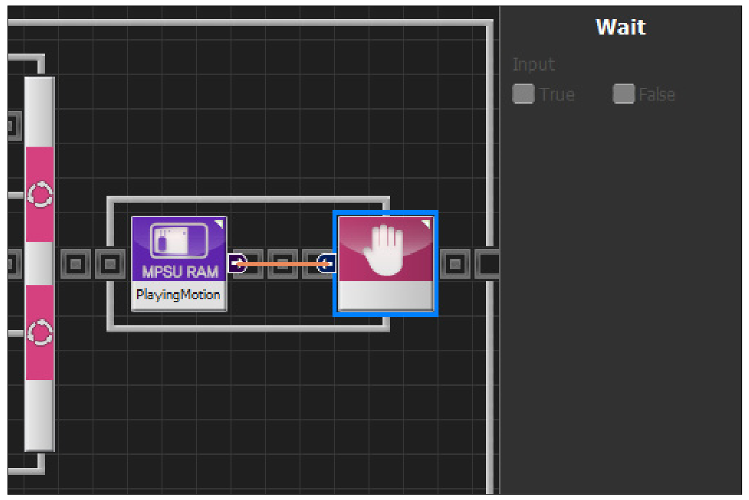


15 Run Left Turn Motion

Add Move module for running motion # 2 (Left turn) to the second program line of the If-Else module. Robot will make a left turn if the distance value is less than 20cm.

Select Motion > Move and place the module on the second program line of the If-Else module.

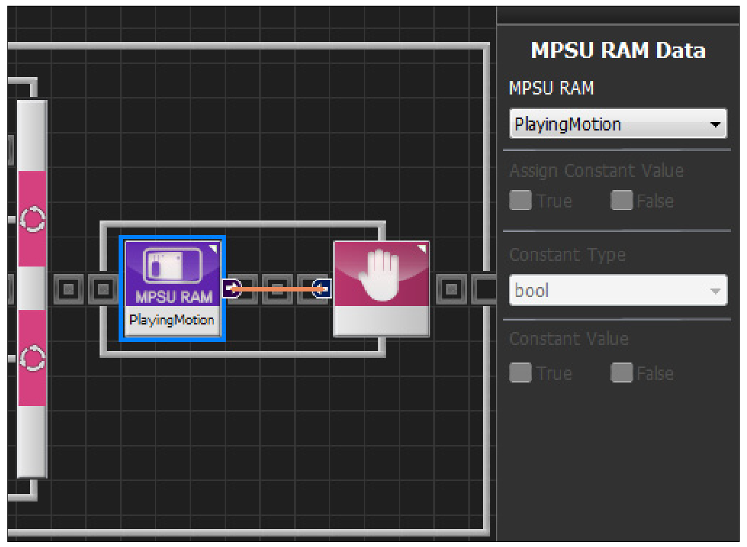
Select Play/Stop : Play
Select Motion Index : 2. Play motion # 2 (left turn motion).
Select Motion Ready : False.



16 Add Wait

Add Wait module after the If-Else module to wait until the motion in the IF-Else module ends.

Select Flow > Wait and place the module after the If-Else module.

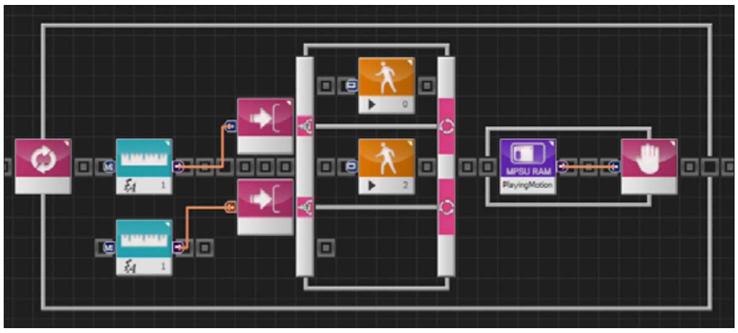


17 Add PlayingMotion Module

Data > MPSU RAM contains "PlayingMotion". This register is a variable which has value of 1 if motion is in progress and value of 0 if motion is not active. Connect 'PlayingMotion' to Wait module input pin to pause the program at the Wait module while the motion is playing.

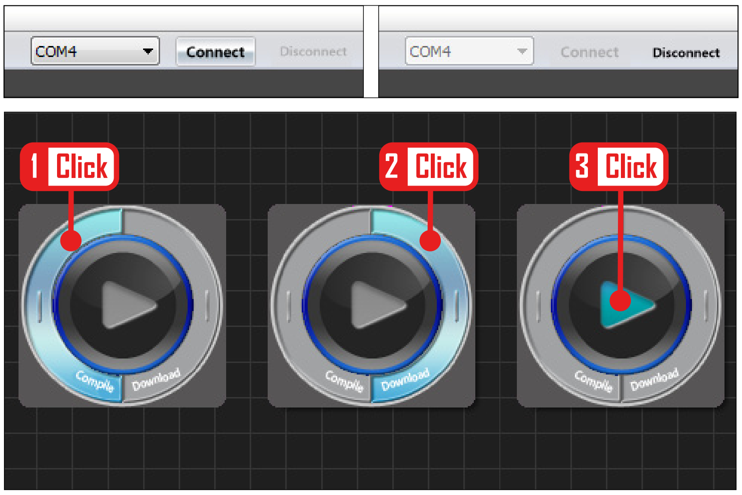
Select Data > MPSU RAM and place the module inside the Wait module.

Select MPSU RAM : Palyingmotion.
Connect the output pin of the module to the input pin of the Wait module.



18 Summary

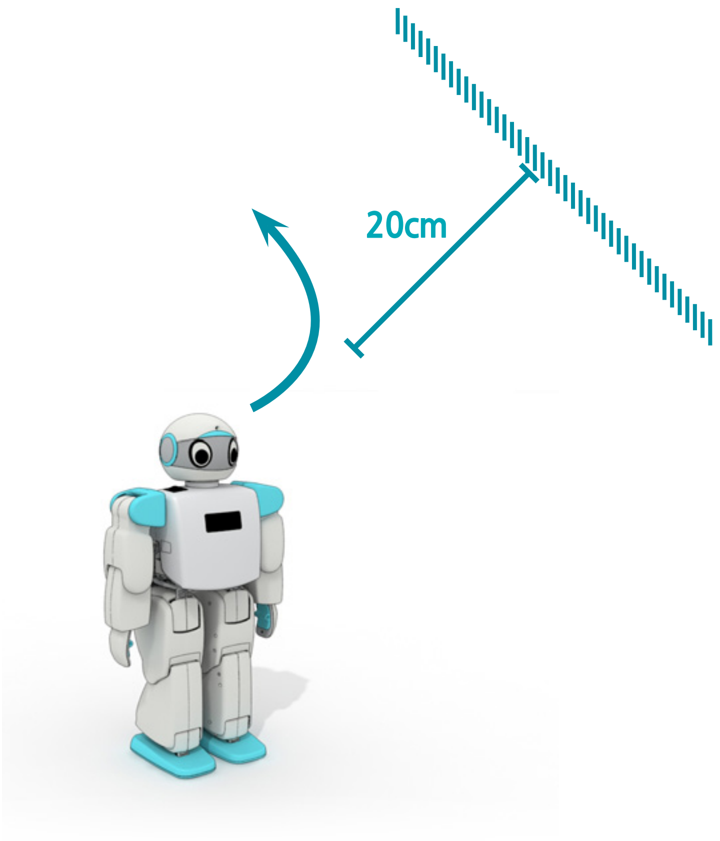
This is the shape of the entire Loop statement. Forward motion will be run if distance sensor value is greater than 20cm and if distance sensor value is less than 20cm, left turn motion will be run. No motion will take place if both are False (when sensor is not connected). After If-Else statement ends, program will pause using 'Wait' until the motion ends. Program will then go back to the beginning of the Loop statement to enter the If-Else statement again depending on the value of the distance sensor.



19 Download

After programming is completed.
Compile -> Download -> Run.

Select the appropriate COMPORT or USB-to-Serial number located on the PC and press Connect to open the serial port. Click 'Compile' and then click 'Download' on the right to download the program to the robot if no compile error is found. Click "Run" button (arrow in the middle) after completing the download.



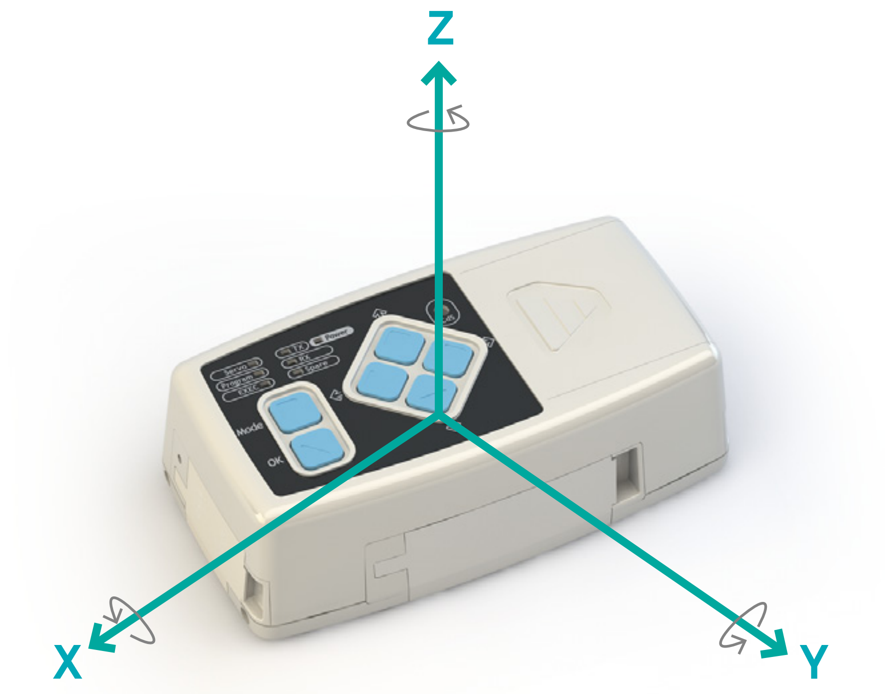
20 Robot Motion

Robot will walk forward and then make a left turn if it detects an obstacle within 20cm distance.

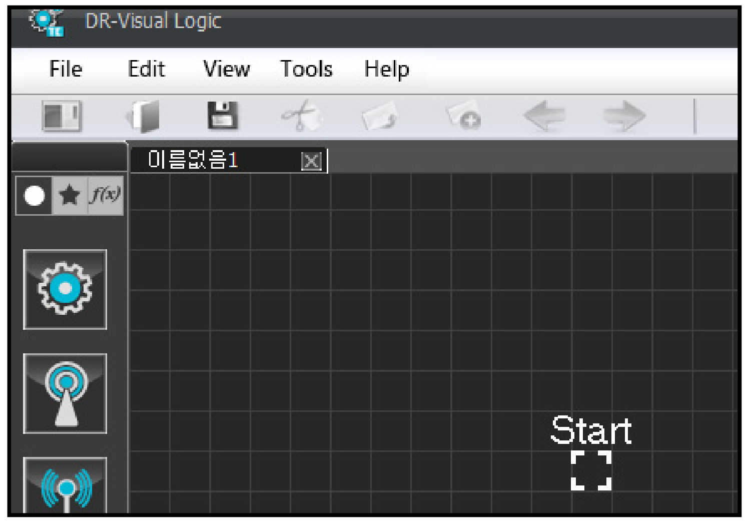
Acceleration

Example Description

This program will make use of the Accelerometer to make the robot get back on its feet after it falls forward and backward. Accelerometer and the Gyro sensor is built as a single module which is inserted into the controller by the back cover.

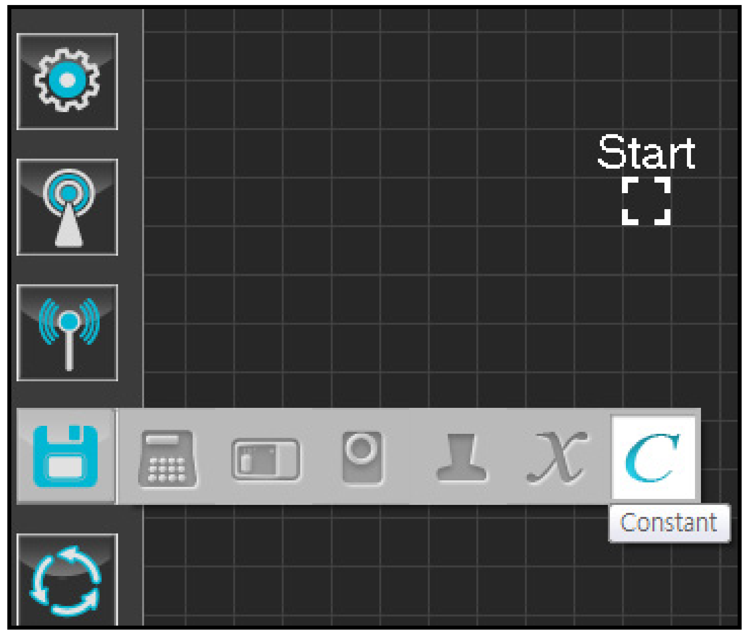


- When the robot is in prone position (lying face down),
  Z axis “+” acclerates and it’s value is approximately +256.
- When the robot is in supine position (lying on the back),
  Z axix “-” accelerates and it’s value is approximately -256. (256 is approximately 1g force of gravity value.)



01 Create New File

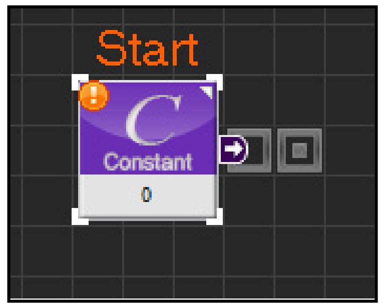
Click on the left most icon in the tools to create new dts file. Select HOVIS Lite/ Eco as the Robot.



02 Select Module

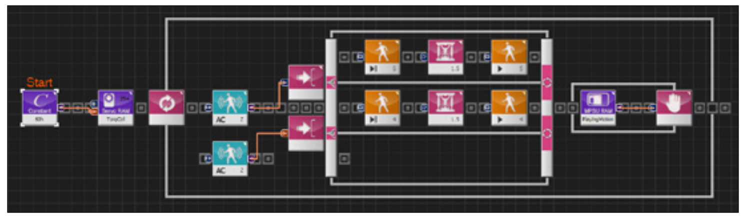
Prior to placing the module, click to select the module from the module bar located on the left side.

Click Data > Constant module



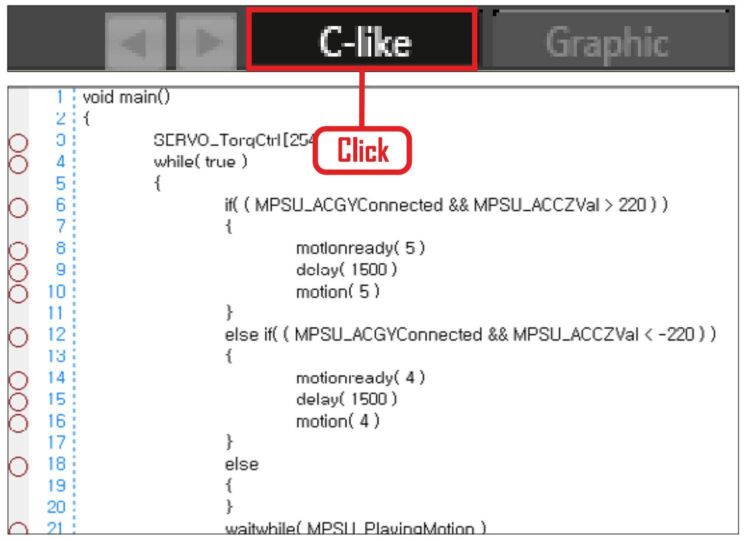
03 Place Module

Drag and dock the module to the Start Point to activate the module. Active module will turn to color.



04 Entire Program

View of the entire program

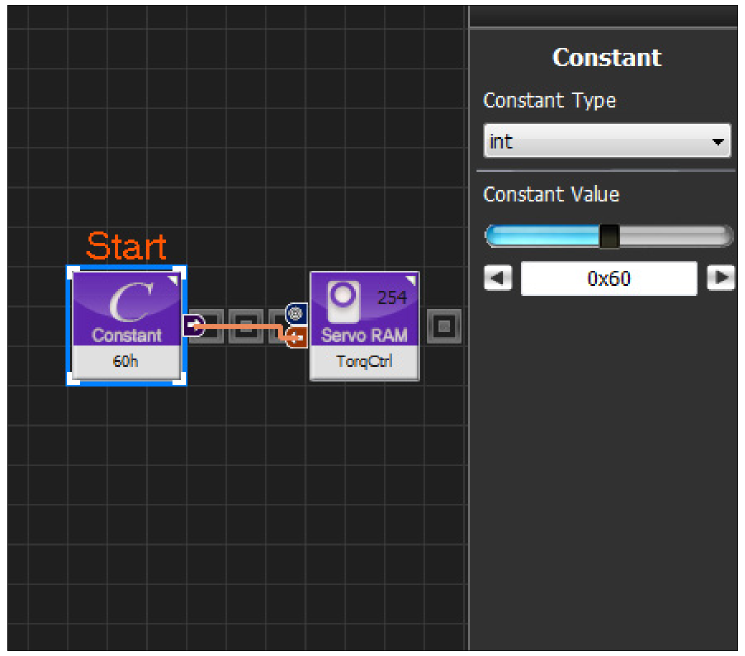


```
void main()
{
        SERVO_TorqCtrl[254
        while( true )
        {
                if( ( MPSU_ACGYConnected && MPSU_ACCZVal > 220 ) )
                {
                        motionready( 5 )
                        delay( 1500 )
                        motion( 5 )
                }
                else if( ( MPSU_ACGYConnected && MPSU_ACCZVal < -220 ) )
                {
                        motionready( 4 )
                        delay( 1500 )
                        motion( 4 )
                }
                else
                {
                }
                waitwhile( MPSU_PlayingMotion )
```

05 View C-Like

Click "C-Like" tab near the top right to open the program source code window as shown in the left.

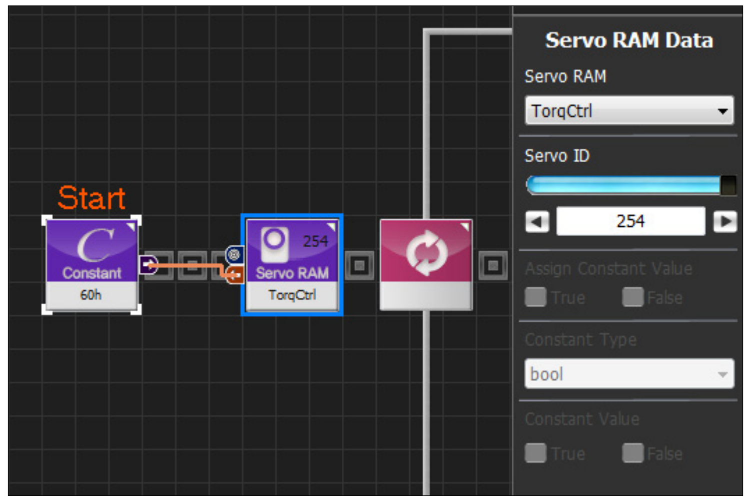
Since codes are very similar to the C language structure, studying the codes will help the user become familiar with the C language as well. Clicking on the module results in cursor jumping to the matching source code line in the C-Like window. This feature enables the user to easily see the codes generated by the module.



Constant
Constant Type: int
Constant Value: 0x60

06 Set Constant

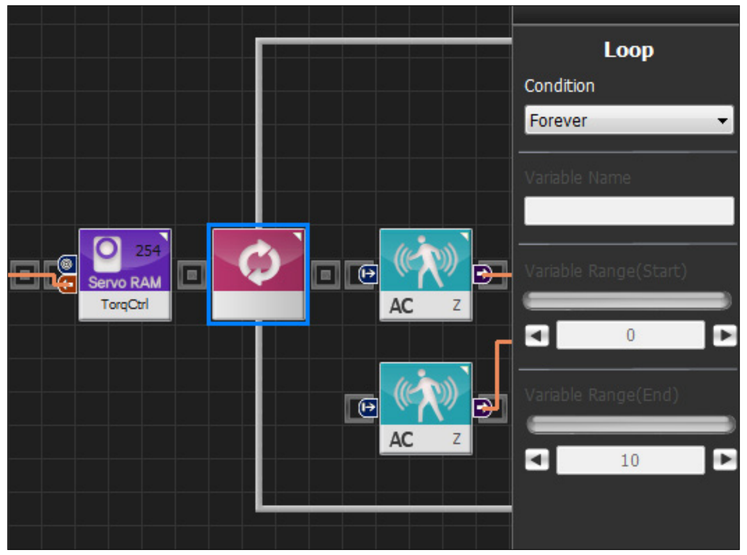
This section allows the servo motor to operate on its own. From the Constant module properties, click on the Constant Value and change the value to 0x60. Constant with Hex number of 0x60 shows the torque enabled state of the servo motor. This value is sent to the input pin of the next module through the output pin



Servo RAM Data
Servo RAM: TorqCtrl
Servo ID: 254
Assign Constant Value: True False
Constant Type: bool
Constant Value: True False

07 Apply to All Servos

This section applies constant value 0x60 received from the previous section to all servo motors.
Select Data > Servo RAM and place it after the Constant module.
Servo RAM : Select TorqCtrl
Servo ID : Set to 254, ID 254 means setup will be applied to all connected servos.
Use the Connector to connect the output pin of the Constant module to the second input pin of the Servo RAM module.

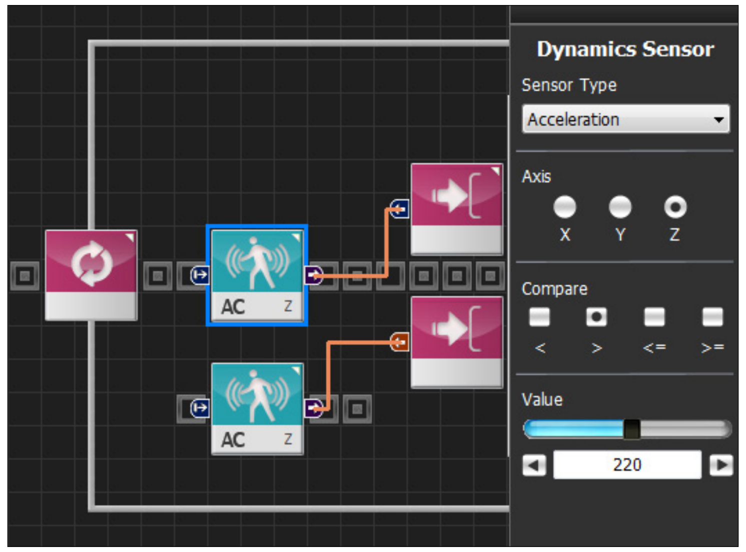


Loop
Condition: Forever
Variable Name
Variable Range(Start): 0
Variable Range(End): 10

08 Loop

Place loop module and set it to infinite loop mode to continuously repeat the motion. Content of the loop module will be repeated continuously.

Select Flow > Loop and place it after the previous module.
Set Condition : Forever, for infinite loop.

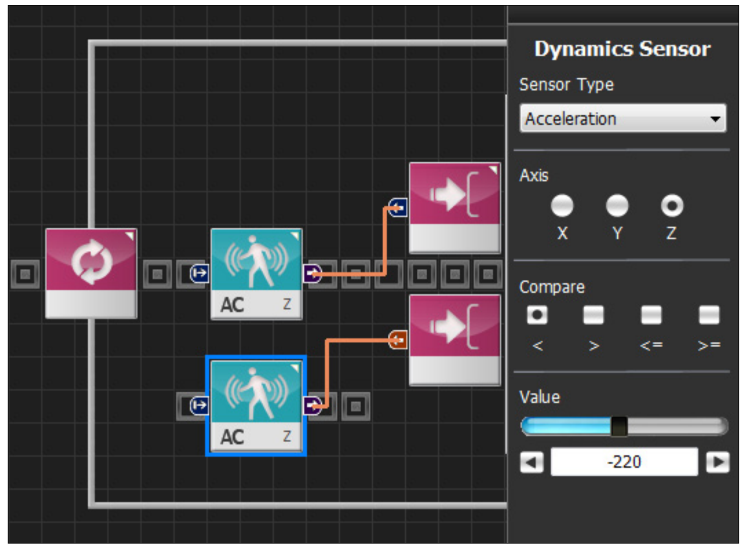


Dynamics Sensor
Sensor Type: Acceleration
Axis: X Y Z
Compare: < > <= >=
Value: 220

09 Add Accelerometer

Add Accelerometer module inside the loop statement. Z axis value is 0 when robot is standing straight, +256 when robot is lying face down (prone position), and -256 when lying on its back (supine position). Therefore, when Z axis value approaches 256, it's safe to assume that robot is lying face down on the ground. Set reference value as +220.
This module is connected to the Accelerometer/Gyro sensor module and outputs True/False after using 'Compare' to compare the value with 'Value'. Output is 'False' if sensor is not connected.

Select Sensor > Dynamics and place the module inside the Loop module.
Select Sensor Type
: Acceleration (Accelerometer)
Set Axis : Z axis
Set Compare : >. Output True when sensor value is greater than 'Value'.
Set Value : +220

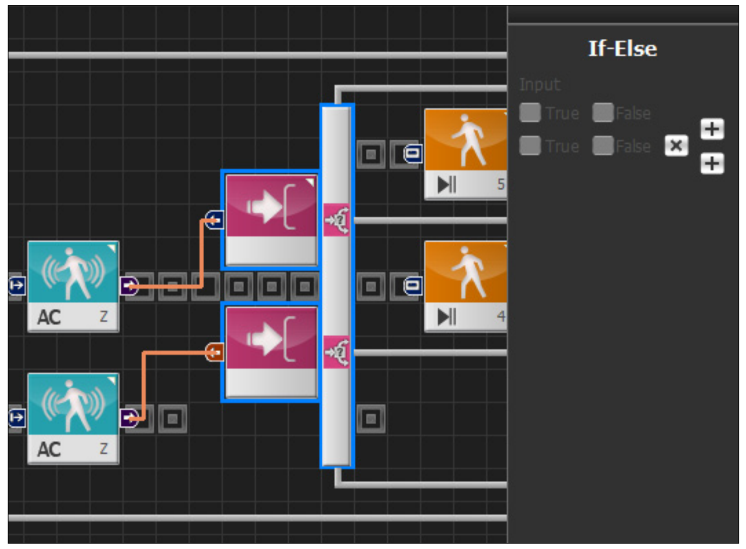


Dynamics Sensor
Sensor Type: Acceleration
Axis: X Y Z
Compare: < > <= >=
Value: -220

10 Add Accelerometer

Add one more Accelerometer module and place it below the module added in step 11. Unlike the previous module, Compare will be set to < and Value to -220. Value of this module becomes close to -220 when robot falls backward and lies on its back (supine position).

Select Sensor > Dynamics and place the module inside the Loop module.
Select Sensor Type
: Acceleration (Accelerometer)
Set Axis : Z axis
Set Compare : <. Output True when sensor value is less than 'Value'.
Set Value : -220

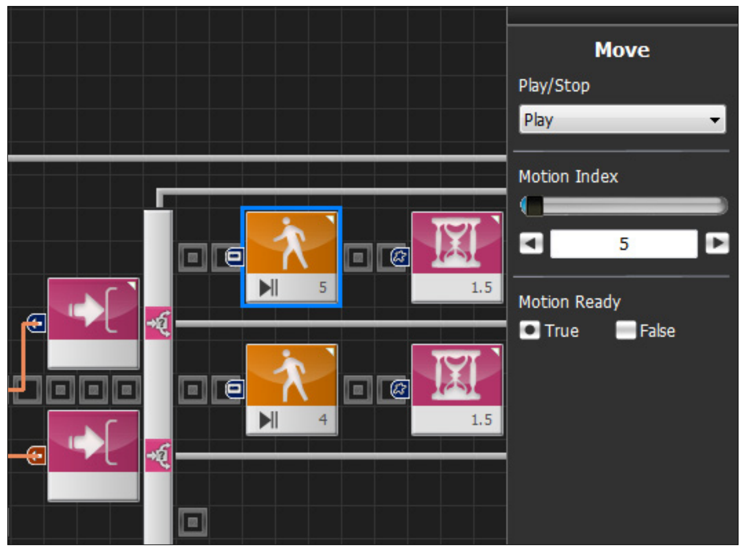


If-Else
Input: True False
True False

11 Create If-Else Statement

Create If-Else statement and connect it to the previously made sensor module.

Select Flow > If-Else and place the module within the Loop module.
Add one more conditional statement by clicking + shaped icon.
Connect the Accelerometer module output pin from step 9 to the first input pin and output pin from step 10 to the second input pin.


Move
Play/Stop
Play
Motion Index
5
Motion Ready
True False

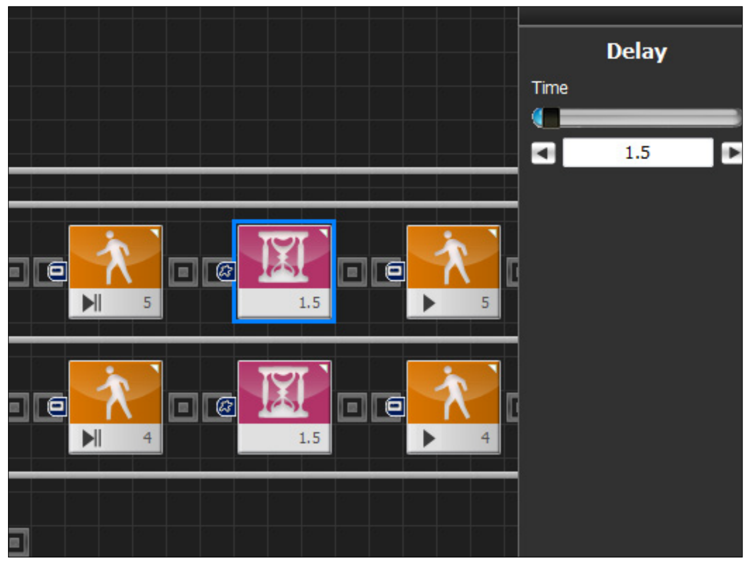
12 Motion Ready

Fist program line of the IF-Else statement runs when robot is in prone position.

When motion is first loaded, robot may make a sudden movement. If the difference between the current position and the start of the motion is very different, it may cause stress to the motor or pose danger to the user. To prevent such an occurrence, motion is run in 'Motion Ready' mode to provide time for motion to start.

Select Motion > Move module and place it in the first line of the If-Else statement.
Play/Stop : Select Play.
Select Motion Index : 5, load motion number 5 which makes the robot get up from the face down position.

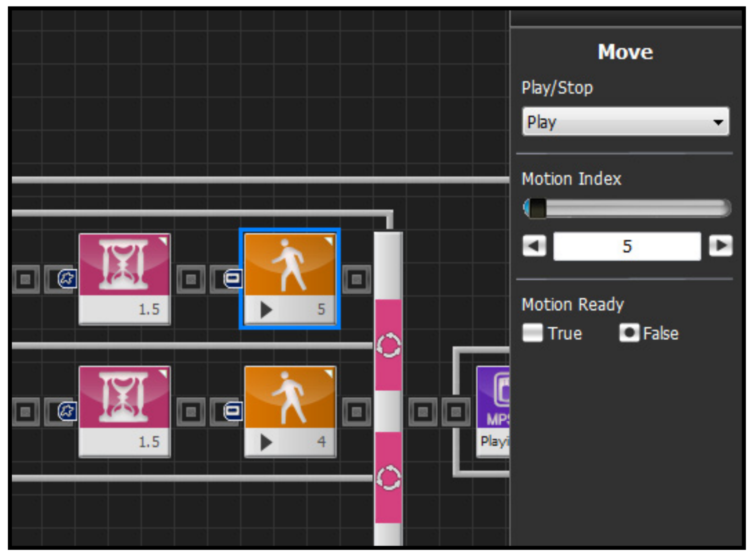
Select Motion Ready : True. When set to True, robot will slowly move to starting position of the motion.


Delay
Time
1.5

13 Delay

Set Delay value to 1.5s to prevent robot motion from starting before the Motion Ready ends.

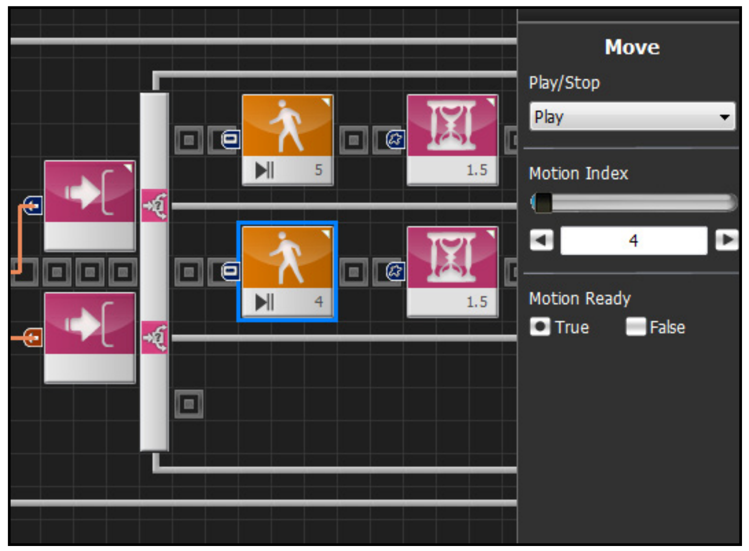
Select Flow > Delay module and place it after the previous module.
Set Time: 1.5. Delay 1.5s.


Move
Play/Stop
Play
Motion Index
5
Motion Ready
True False

14 Run Up Motion

Add Move module to run motion #5 (up from prone position).

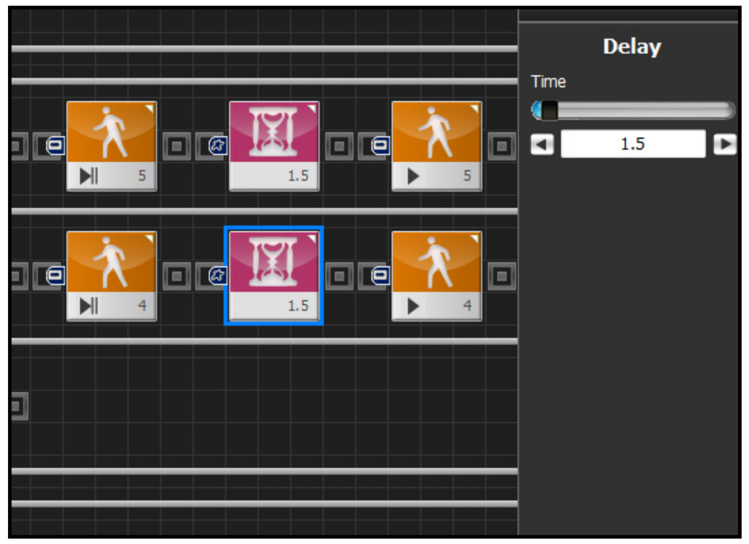
Select Motion > Move and place the module after the previous module.
Select Play/Stop : Play
Select Motion Index : 5. Play motion # 5 (up motion from prone position).
Select Motion Ready : False.


Move
Play/Stop
Play
Motion Index
4
Motion Ready
True False

15 Motion Ready

Second line in the If-Else statement is run when robot is in supine position (lying on its back). Motion #4 makes the robot get back on its feet from the supine position. Again, Motion Ready is used to slowly ready the robot for the motion.

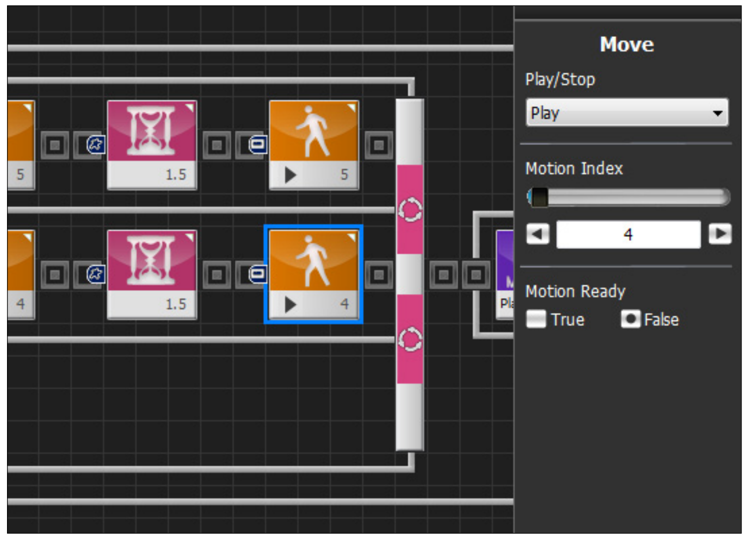
Select Motion > Move and place the module after the previous module.
Select Play/Stop : Play
Select Motion Index : 4. Play motion # 4 (up motion from supine position).
Select Motion Ready : False.


Delay
Time
1.5

16 Delay

Set Delay value to 1.5s to prevent robot motion from starting before the Motion Ready ends.

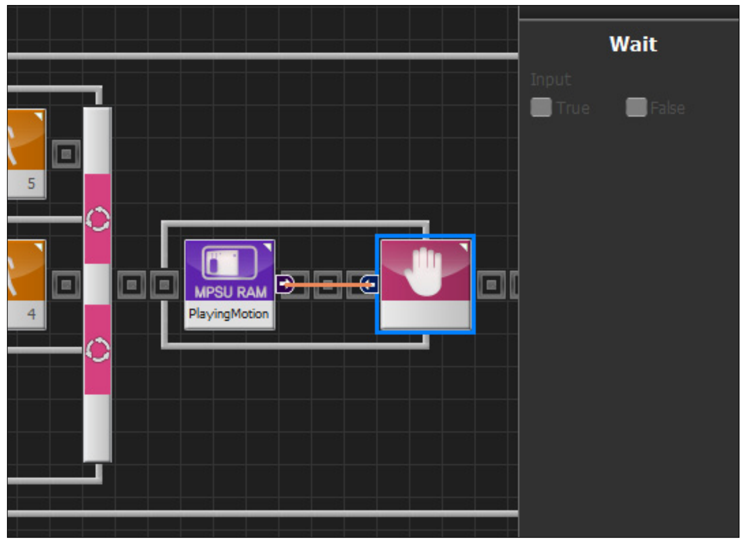
Select Flow > Delay module and place it after the previous module.
Set Time: 1.5. Delay 1.5s.



17 Run Up Motion

Add Move module to run motion #4 (up from prone position).

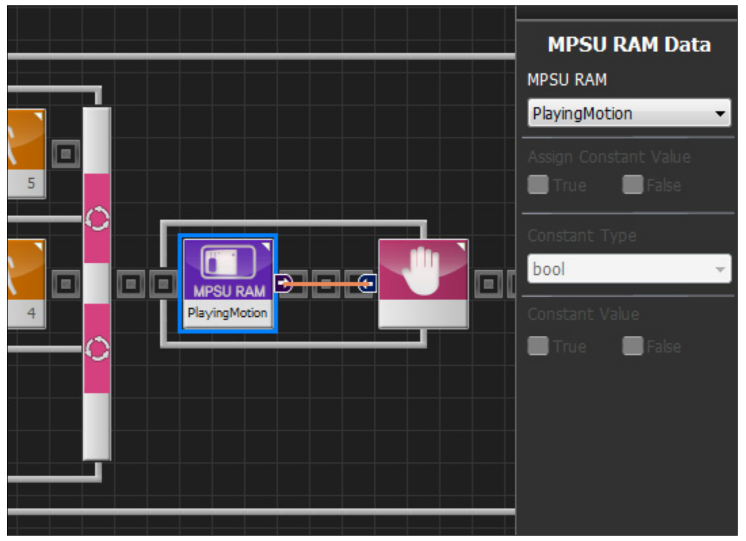
Select Motion > Move and place the module after the previous module.
Select Play/Stop : Play
Select Motion Index : 4. Play motion # 4 (up motion from supine position).
Select Motion Ready : False.



18 Add Wait.

Add Wait module after the If-Else module to wait until the motion in the IF-Else module ends.

Select Flow > Wait and place the module after the If-Else module.

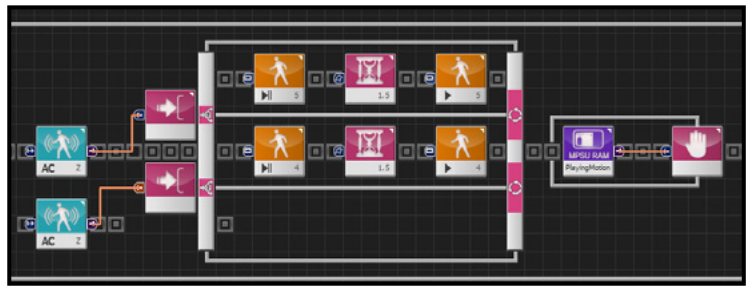


19 Add PlayingMotion Module

Data > MPSU RAM contains "PlayingMotion". This register is a variable which has value of 1 if motion is in progress and value of 0 if motion is not active. Connect 'PlayingMotion' to Wait module input pin to pause the program at the Wait module while the motion is playing.

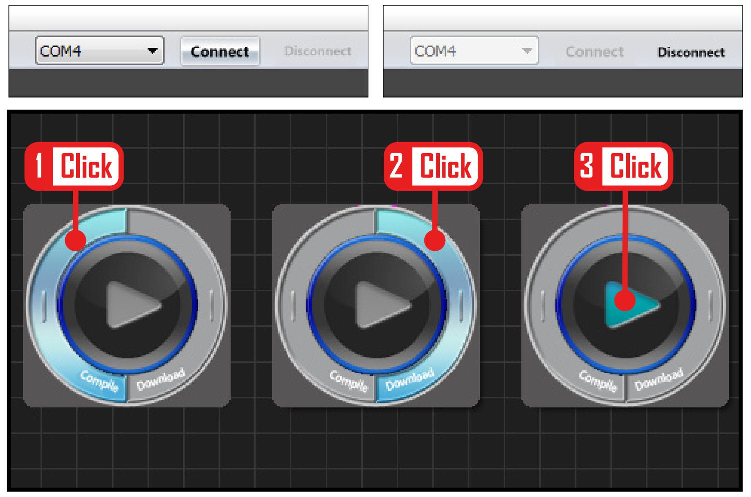
Select Data > MPSU RAM and place the module inside the Wait module.

Select MPSU RAM
: Palyingmotion. Connect the output pin of the module to the input pin of the Wait module.



22 Summary

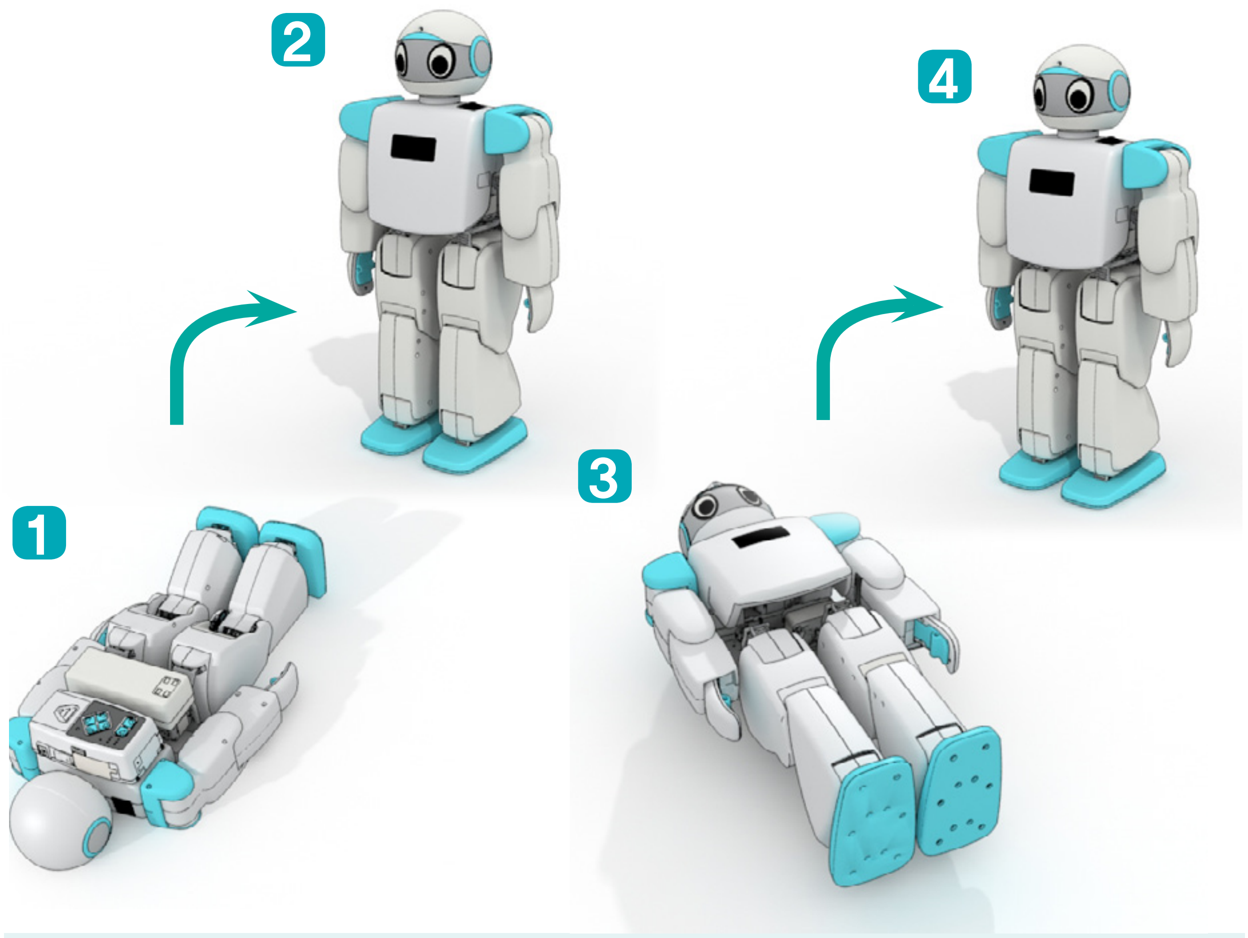
This is the shape of the entire Loop statement. If robot is in prone position (lying face down), Motion Ready for motion #5 will be run first and then motion #5 would be run after 1.5s delay. Motion # 4 would be run if robot is in supine position. No motion will take place if both are False (when robot is in standing position). After If-Else statement ends, program will pause using 'Wait' until the motion ends and then go back to the beginning of the Loop statement to enter the If-Else statement again depending on the value of the Accelerometer.



23 Download

After programming is completed.
Compile -> Download -> Run.

Select the appropriate COMPORT or USB-to-Serial number located on the PC and press Connect to open the serial port.
Click 'Compile' and then click 'Download' on the right to download the program to the robot if no compile error is found.
Click "Run" button (arrow in the middle) after completing the download.



24 Robot Motion

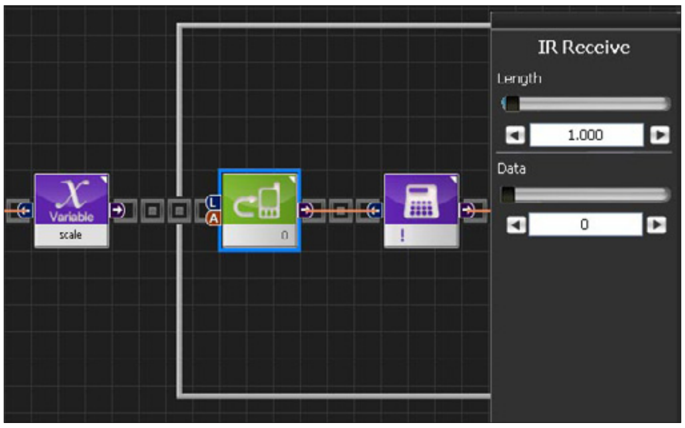
If the robot is in prone position, it gets back up backwards. If it is in supine position, it gets back up forward.

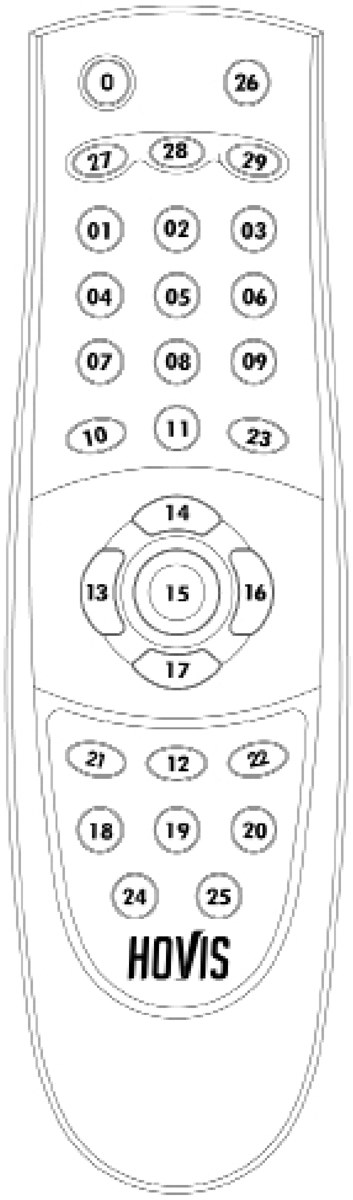
IR Receive, Sound & Motion

Example Description

(Explain by Sound examples, skip the explaination of motion examples, Data Match for Remote Controller)

Data figure from IR Recieve Module shall match the key from on the right side of remote controller.



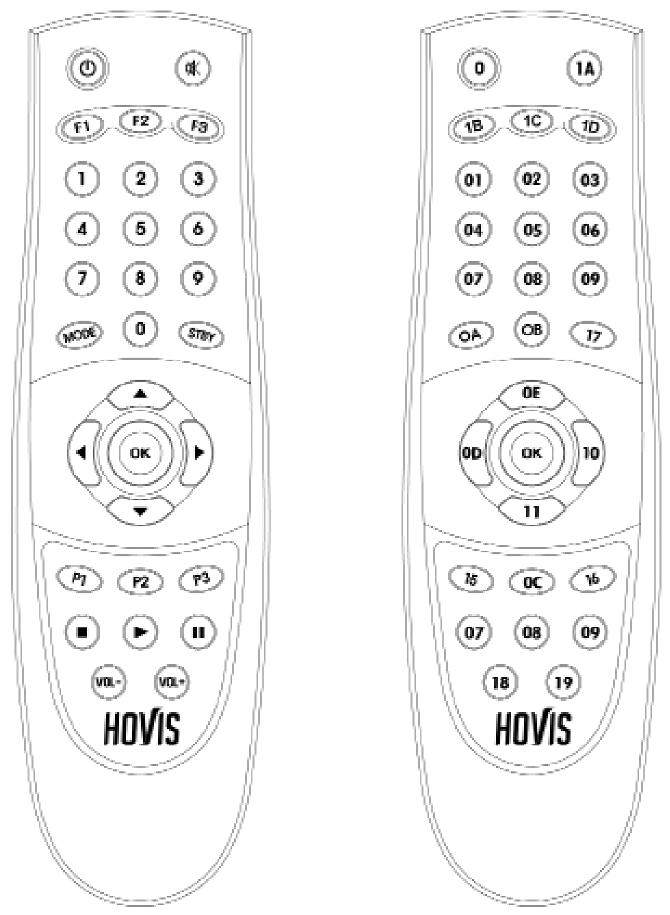


Hovis remote control keymap is as shown in the picture to the right. IR Receive module data values correspond to numbers in the right key.

For example, if the top right power button is pressed, Data 0 is received by the DRC. Robot can be programmed to take certain action whenever the power button is pressed by setting the Data to 0 in the IR RECEIVE module and connecting to If-Else module input

Channel setting

Bothe the Remote control channel and DRC channel is user selectable but selected channel in DRC must match the remote controller channel in order for DRC to receive data from the remote control. Remote control channel can be selected by pressing 1~0 number + OK button simultaneously. DRC channel is selected by changing the RmcChannel value in MPSU Ram Data. RmcChannel values corresponding to remote control numbers are as follows.

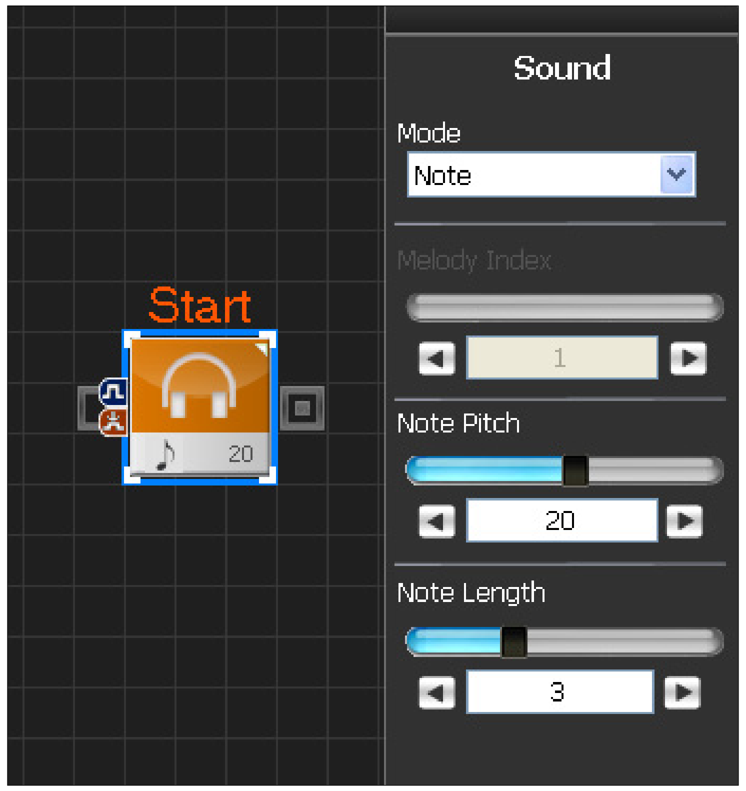


Remote Control Button	RmcChannel Value
0+OK	97(0x61)
1+OK	98(0x62)
2+OK	99(0x63)
3+OK	100(0x64)
4+OK	101(0x65)
5+OK	102(0x66)
6+OK	103(0x67)
7+OK	104(0x68)
8+OK	105(0x69)
9+OK	106(0x6A)

Example Description

This example associates remote control number button to a music note and outputs Do,Re,...Do (1~8) notes.

Note pictch is dependent on the value of the Note Pitch in Motion > Sound module. DRC controller has total of 38 pitches from 0~37 and it is able to ouptut total of 3 octaves.



Start

20

Sound

Mode: Note

Melody Index: 1

Note Pitch: 20

Note Length: 3

00 Sound Property Window

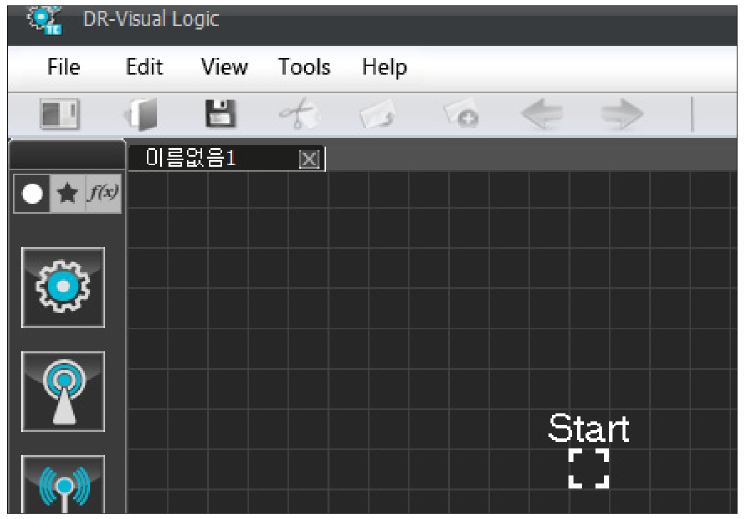
Select Motion > Sound module.
Mode has Melody & Note. Melody selects and plays one of the saved edited notes.
Note Mode is selected to use the 36 note pitches.
Refer to the table below
Note Pitch from 0~37 can be selected.
Note pitches comprise total of 3 octaves.
Note Length refers to the beat.
Thirty-second note to the whole note can be selected.
Refer to the table below

Note Pitch

No.	0												
Note	NA												
No.	1	2	3	4	5	6	7	8	9	10	11	12	
Note	Do	Do#	Re	Re#	Mi	Fa	Fa#	Sol	Sol#	La	La#	Si	
No.	13	14	15	16	17	18	19	20	21	22	23	24	
Note	Do	Do#	Re	Re#	Mi	Fa	Fa#	Sol	Sol#	La	La#	Si	
No.	25	26	27	28	29	30	31	32	33	34	35	36	37
Note	Do	Do#	Re	Re#	Mi	Fa	Fa#	Sol	Sol#	La	La#	Si	Do

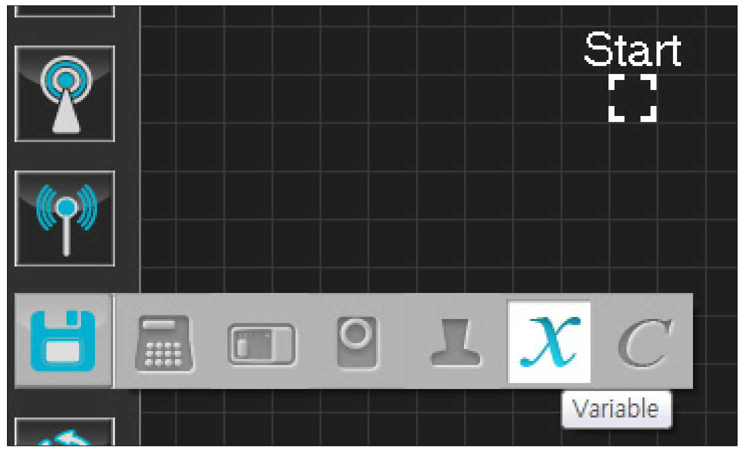
Note Length

No.	0	1	2	3	4	5	6	7	8	9
Raw Data	6	12	18	24	36	48	72	96	144	192
ms	38.4	76.8	115.2	153.6	230.4	307.2	460.8	614.4	921.6	1228.8
Note	32 note	16 note	16 dot note	8 note	8 dot note	4 note	4 dot note	2 note	2 dot note	Whole note



01 Create New File

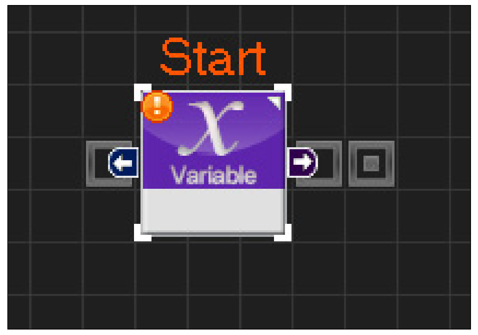
Click on the left most icon in the tools to create new dts file. Select HOVIS Lite/Eco as the Robot.



02 Select Module

Prior to placing the module, click to select the module from the module bar located on the left side.

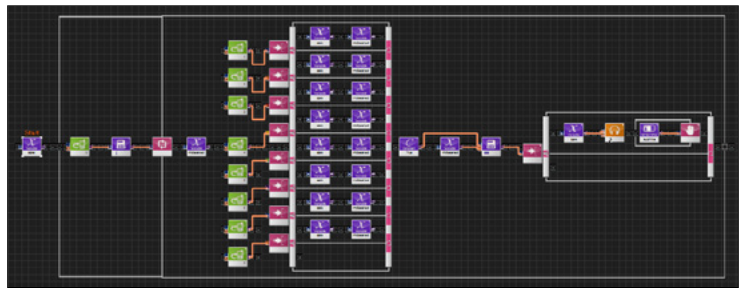
Click Data > Constant module



03 Place Module

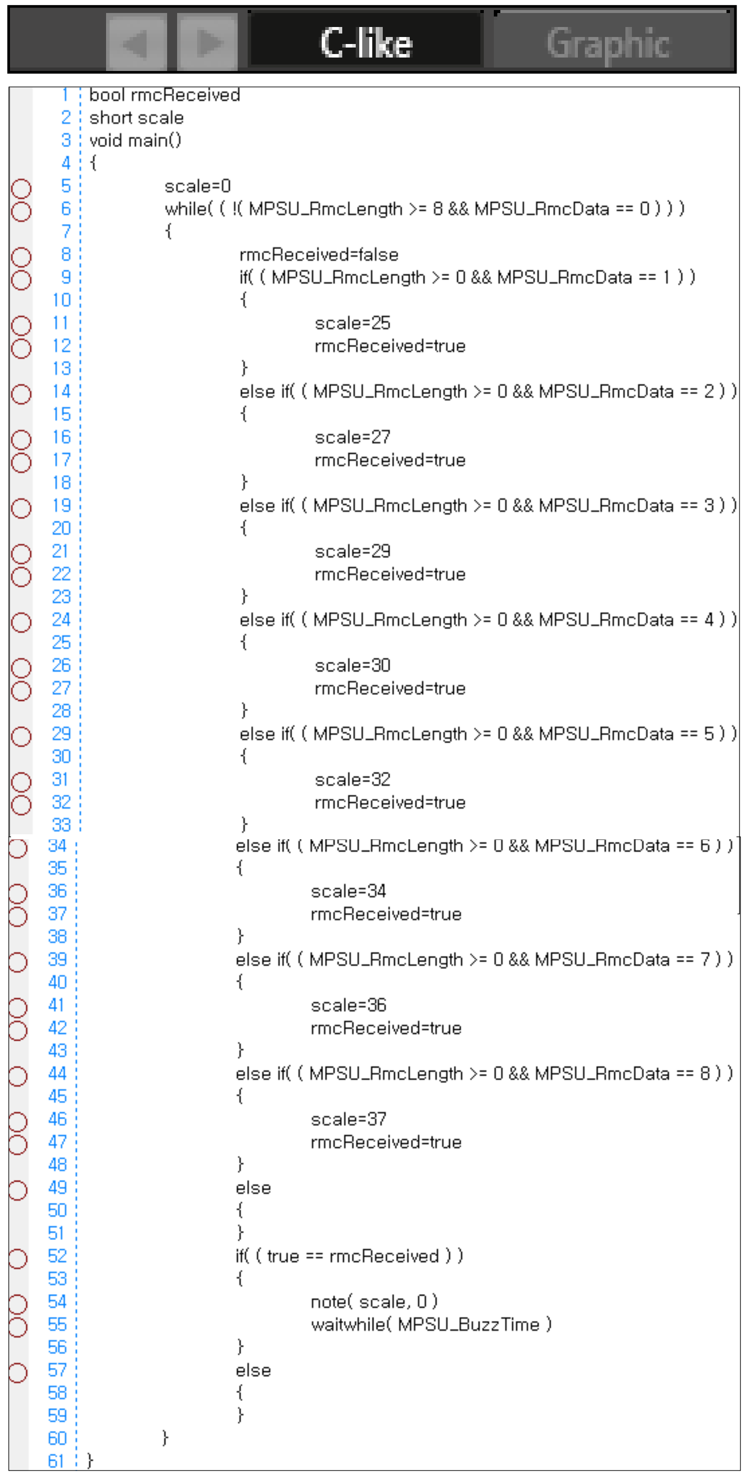
Drag and dock the module to the Start Point to activate the module.

Active module will turn to color.



04 Entire Program

View of the entire program

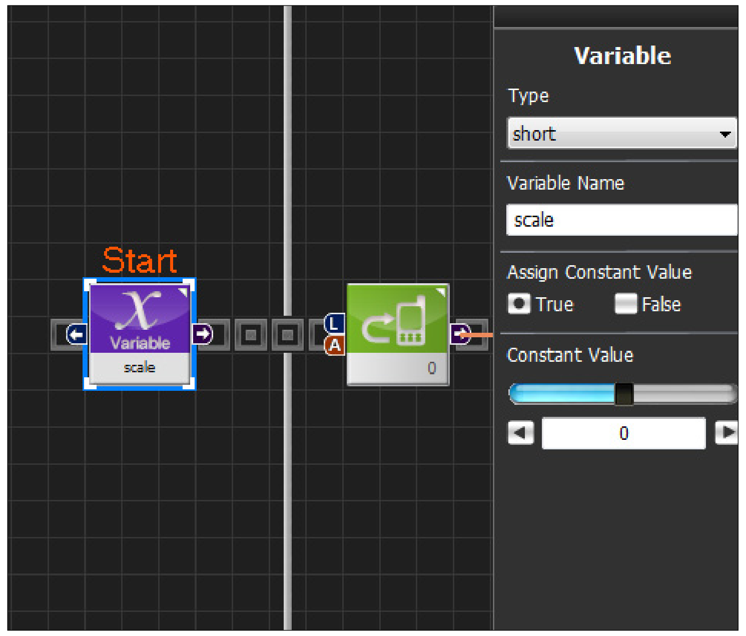


```
bool rmcReceived
short scale
void main()
{
        scale=0
        while( ( !( MPSU_RmcLength >= 8 && MPSU_RmcData == 0 ) ) )
        {
                rmcReceived=false
                if( ( MPSU_RmcLength >= 0 && MPSU_RmcData == 1 ) )
                {
                        scale=25
                        rmcReceived=true
                }
                else if( ( MPSU_RmcLength >= 0 && MPSU_RmcData == 2 ) )
                {
                        scale=27
                        rmcReceived=true
                }
                else if( ( MPSU_RmcLength >= 0 && MPSU_RmcData == 3 ) )
                {
                        scale=29
                        rmcReceived=true
                }
                else if( ( MPSU_RmcLength >= 0 && MPSU_RmcData == 4 ) )
                {
                        scale=30
                        rmcReceived=true
                }
                else if( ( MPSU_RmcLength >= 0 && MPSU_RmcData == 5 ) )
                {
                        scale=32
                        rmcReceived=true
                }
                else if( ( MPSU_RmcLength >= 0 && MPSU_RmcData == 6 ) )
                {
                        scale=34
                        rmcReceived=true
                }
                else if( ( MPSU_RmcLength >= 0 && MPSU_RmcData == 7 ) )
                {
                        scale=36
                        rmcReceived=true
                }
                else if( ( MPSU_RmcLength >= 0 && MPSU_RmcData == 8 ) )
                {
                        scale=37
                        rmcReceived=true
                }
                else
                {
                }
                if( ( true == rmcReceived ) )
                {
                        note( scale, 0 )
                        waitwhile( MPSU_BuzzTime )
                }
                else
                {
                }
        }
}
```

05 Viewing C-Like

Click “C-Like” tab near the top right to open the program source code window as shown in the left.

Since codes are very similar to the C language structure, studying the codes will help the user become familiar with the C language as well. Clicking on the module results in cursor jumping to the matching source code line in the C-Like window. This feature enables the user to easily see the codes generated by the module.



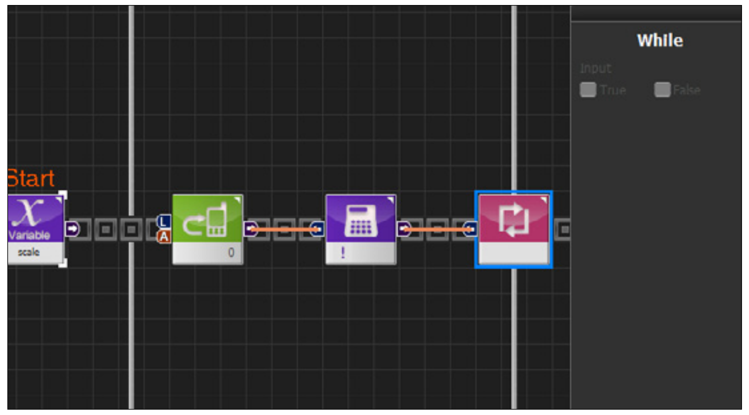
Variable
Type: short
Variable Name: scale
Assign Constant Value: True False
Constant Value: 0

06 Initialize Variable.

Declare variable for use in the program. Declare integer variable 'Scale' to save the pitch of the buzzer sounds to be played. Since this is an integer variable, variable type is 'short'. Up to now, separate constant module was created to assign value to the variable but it is also possible to use only the variable module to assign the value. Set Assign Constant Value to True and then enter the desired constant value in the Constant Value to assign the entered constant value to the variable.

Setup variable module placed at the start point.

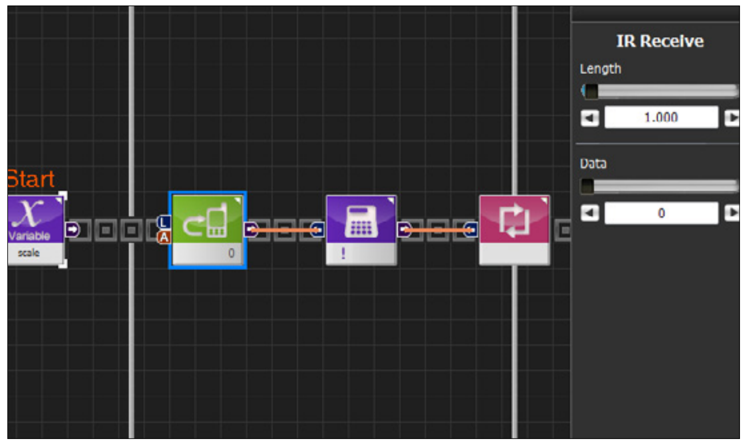
Select Type : Short
Enter Variable Name : Scale
Assign Constant Value : True
Since Constant Value is True, 0 will be assigned to variable 'Scale'.



07 Create While Statement

Similar to the Loop module, While module is also used to repeat the loop statement. Loop module is used to repeat the loop statement continuously or for specified number of times whereas While module is used to repeat the loop as long as the conditional statement set by the user is true.

Select Flow > While and place the module after the Variable module.



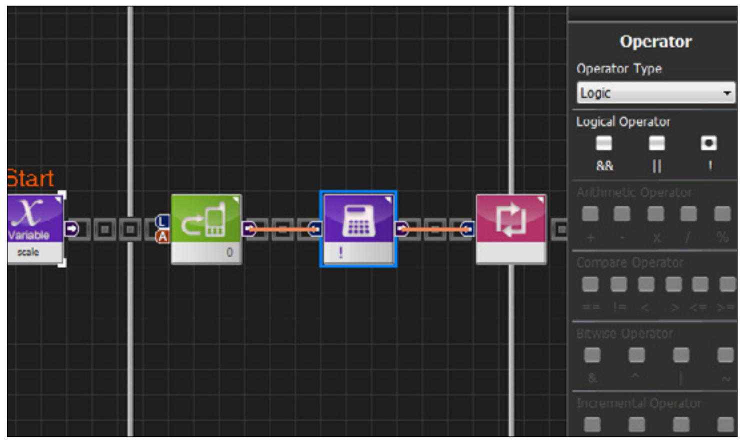
IR Receive
Length: 1.000
Data: 0

08 Add IRReceive Module

Insert the IRReceive module within the left frame of the while statement. IRReceive module has properties named Length and Data. Length refers to the duration of the button press expressed in seconds and Data is the value of the button. When button with the Data key value is pressed for longer than 'Length' seconds, module value becomes True. Otherwise, value becomes False.

Select Communicatin > IRReceive and place the module within the left frame of the While module

Set Length : 1.000
Set Data : 0
Module will output True when power button (key value 0) is pressed for more than 1s.

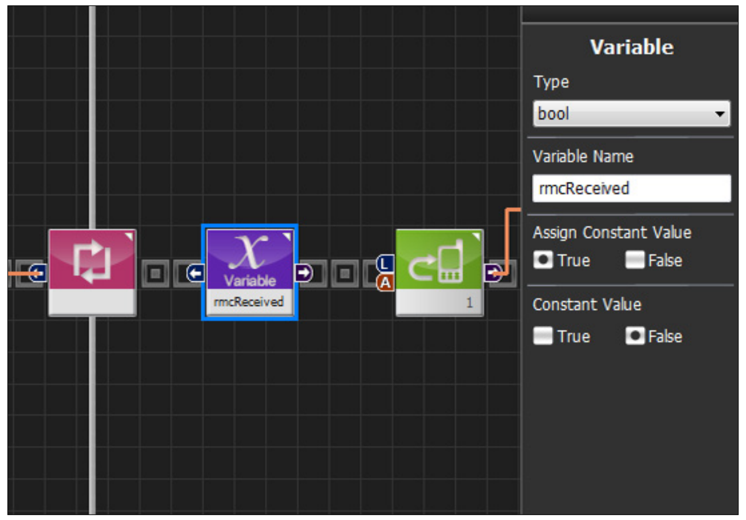


Operator
Operator Type: Logic
Logical Operator: && || !

09 Add Operator

IRReceive added in the previous step can be used as the condition of the While statement to exit the While statement when button is pressed for more than 1s.In order for this to occur, input going into the While statement has to be False when button is pressed for more than 1s and True before the button press. Since this is complete opposite of the output from the remote control module, NOT operator has to be used to reverse the Logical value.

Select Data > Operator and place the module after the IRReceive module.
Select Logical Operator : !(NOT)
Connect the output pin from the IRReceive in step 8 to the input pin of the Operator and connect the output pin from the Operator to the input pin of the While module

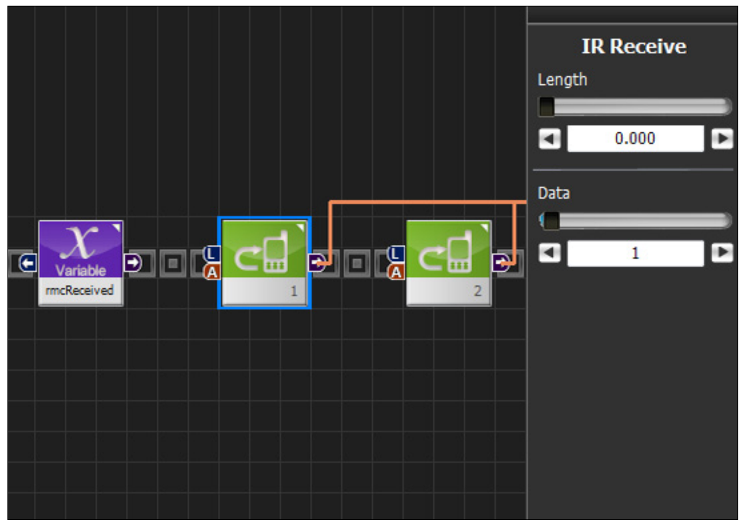


Variable
Type: bool
Variable Name: rmcReceived
Assign Constant Value: True False
Constant Value: True False

10 Initialize Variable

We will start writing the content of the While statement now that it has been programmed to exit from normal continuous loop if power button is pressed for more than 1s. First step is to initialize the variable used for saving remote control input status. Variable is bool type since it only needs to save True/False to express whether input has been received or not. Variable is initialized to False by using the Assign Constant Value option.

Select Data > Variable and place the module inside the While module.
Set Type : bool
Enter Variable Name : rmcReceived
Assign Constant Value : True

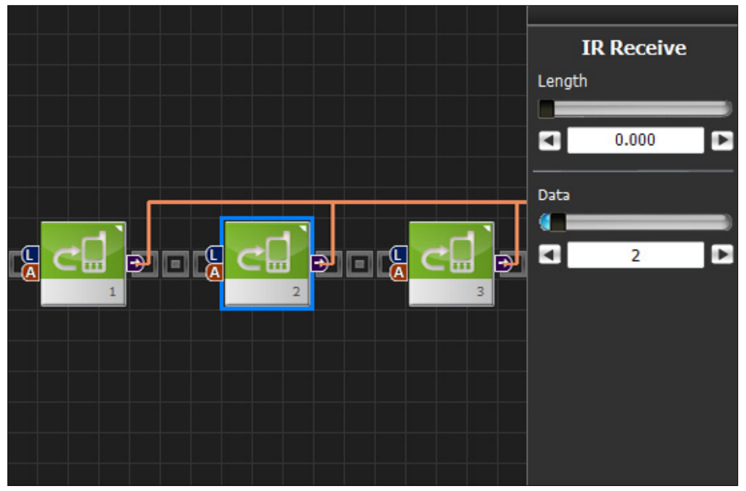

IR Receive
Length
0.000
Data
1

11 Add IRReceive Module (Button 1)

Add IRReceive module and set Length to 0s, and Data to 1. When remote control button 1 is pressed, value becomes 1 regardless of the length of the button press.

Select Communication > IRReceive and place the module after the previous module.

Set Length : 0.000
Set Data : 1

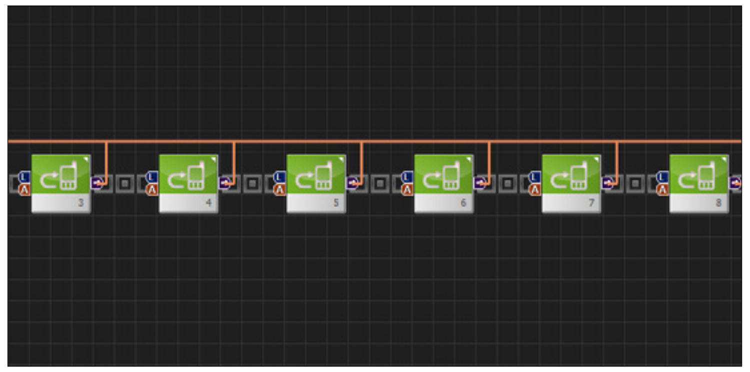

IR Receive
Length
0.000
Data
2

12 Add IRReceive Module (Button 2)

Add IRReceive module and set Length to 0s, and Data to 2.

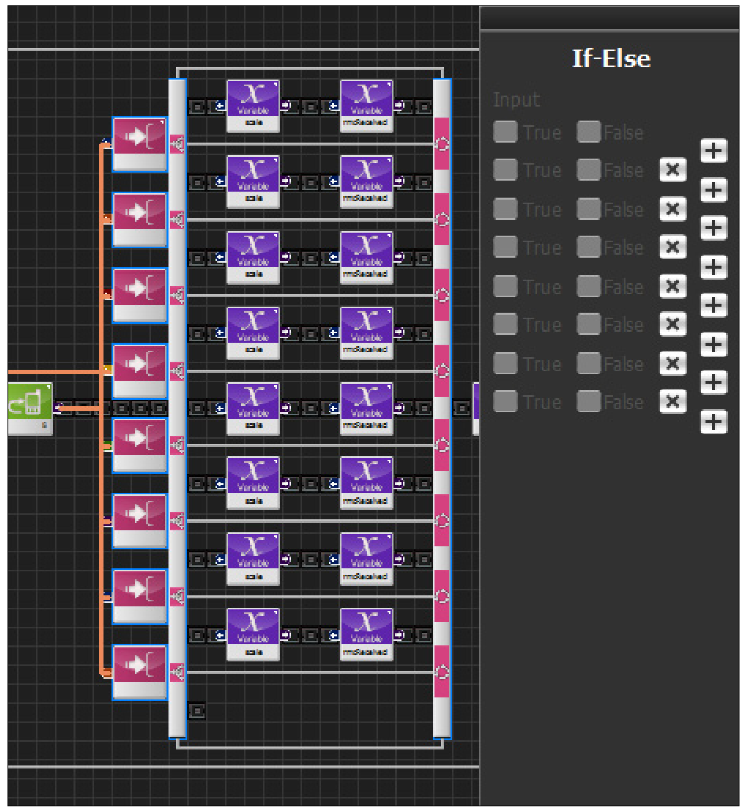
Select Communication > IRReceive and place the module after the previous module.

Set Length : 0.000
Set Data : 2



13 Add IRReceive Modules (Buttons 3~8)

Add IRReceive module (Data : 3~8) for each button number 3 ~ 8 by using same method as shown in steps 11 and 12.

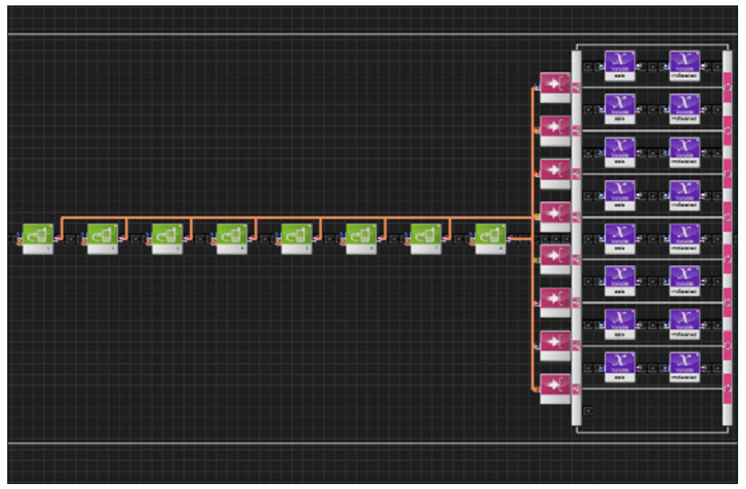


14 Create If-Else Statement

Create If-Else statement and connect the previously made remote control modules to the If-Else statement.

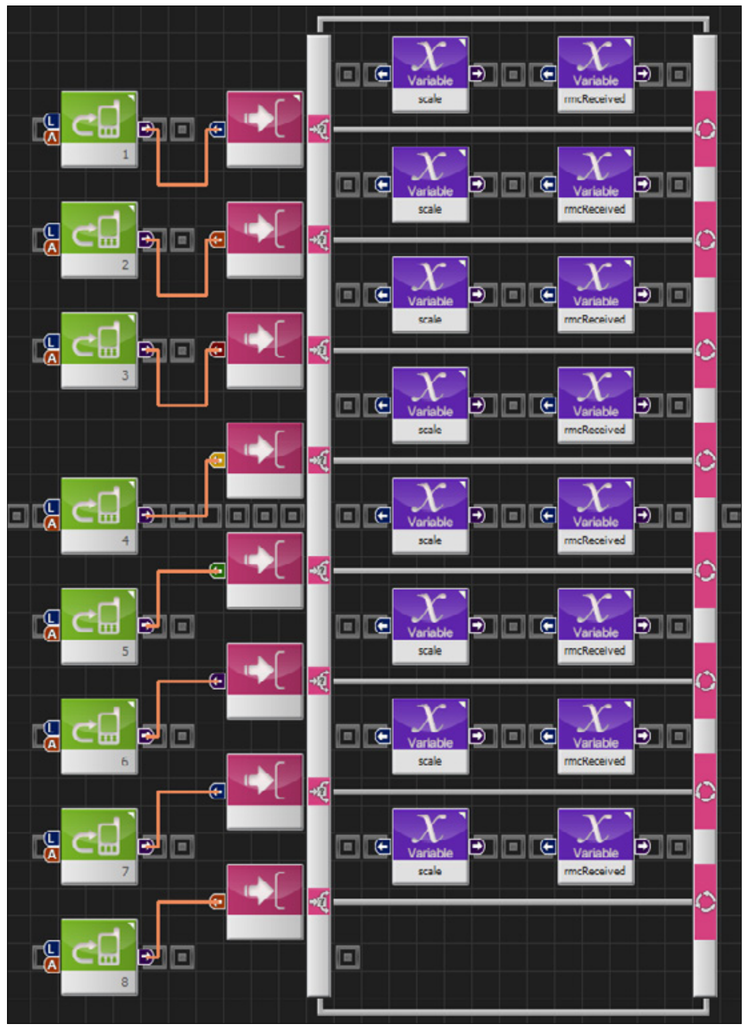
Select Flow > If-Else and place the module after the previously made module.

Click + icon seven times to add 7 more If-Else statements.
Connect output pins from the eight IRReceive modules added in steps 11~13 to the If-Else input pins one by one from 1 to 8.



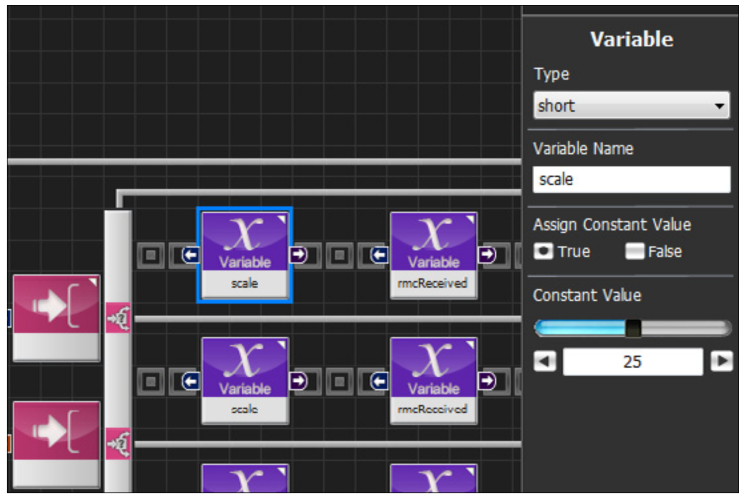
15 Connected If-Else Module

Diagram showing the If-Else Module connected to remote control modules.



16 Module Arrangement

There are 8 If-else modules connected to 8 IRReceive modules. It would be difficult to tell which module was connected to which pin if all connections were placed in the same line. Rearranging the modules by placing IRReceive modules corresponding to buttons 1~3 above the program line and modules corresponding to buttons 5~8 below the program line would make distinguishing each connection much easier.



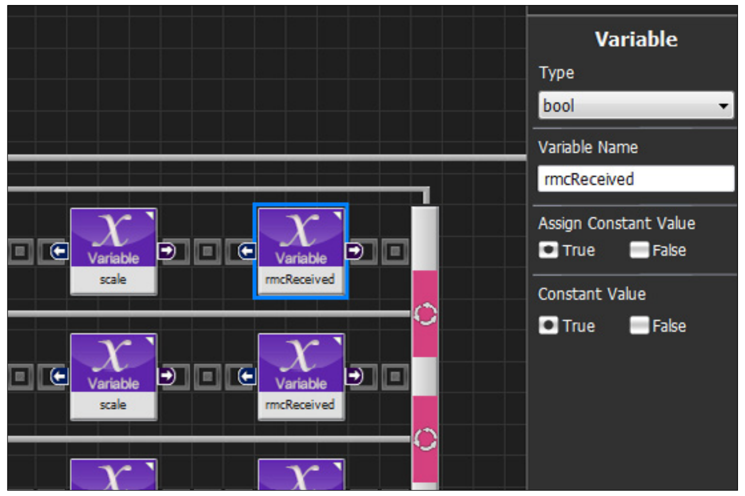
Variable
Type: short
Variable Name: scale
Assign Constant Value: [✓ True] False
Constant Value: 25

17 Save 'Do' In Variable "scale".

First program line in the If-else statement will be executed when button 1 is pressed. 25, which corresponds to 3 Octave "Do" will be assigned to the variable "scale"

Select Data > Variable and place the module in the first program line of the If-Else module.

Select Type : short.
Enter Variable Name : scale
Assign Constant Value : True. Since Assign Constant Value is True, 25 will be assigned to variable "scale".



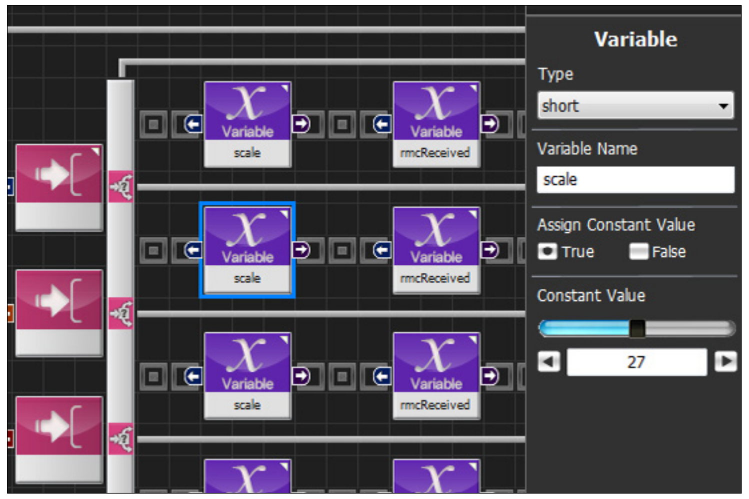
Variable
Type: bool
Variable Name: rmcReceived
Assign Constant Value: [✓ True] False
Constant Value: [✓ True] False

18 Save True In Variable "rmcReceived"

Variable "rmcReceived" was initialized to False at the beginning of the If-Else statement. Assign True to the variable to save remote control signal received state.

Select Data > Variable and place the module after the previous modul.

Set Type : bool
Enter Variable Namue : rmcReceived
Assign Constant Value : True
Select Constant Value : True.



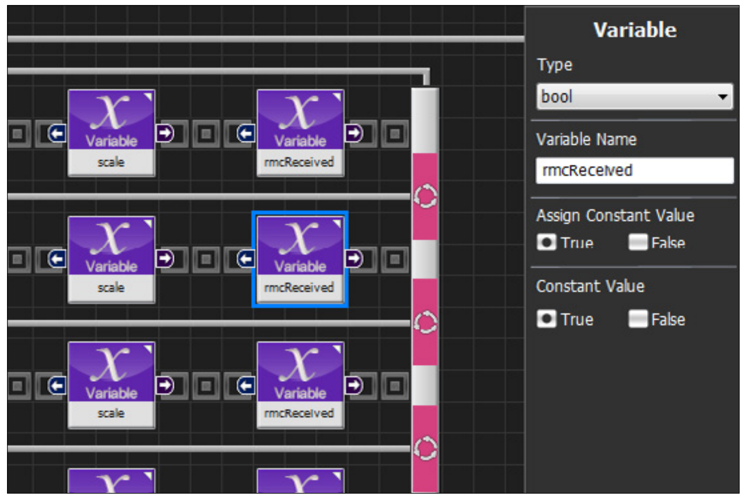
Variable
Type: short
Variable Name: scale
Assign Constant Value: [✓ True] False
Constant Value: 27

19 Save 'Re" In Variable "scale".

Second program line in the If-else statement will be executed when button 2 is pressed. 27, which corresponds to 3 Octave "Re" will be assigned to the variable "scale"

Select Data > Variable and place the module in the second program line of the If-Else module.

Select Type : short.
Enter Variable Name : scale
Assign Constant Value : True. Since Assign Constant Value is True, 27 will be assigned to variable "scale".



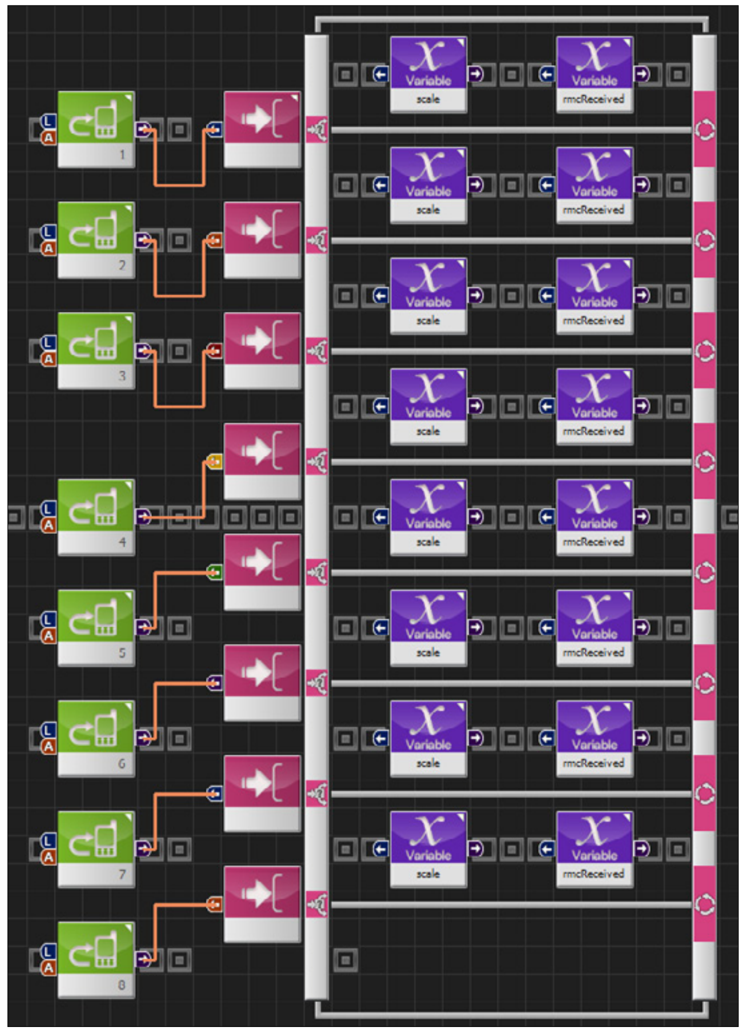
Variable
Type: bool
Variable Name: rmcReceived
Assign Constant Value: [✓ True] False
Constant Value: [✓ True] False

20 Save True In Variable "rmcReceived"

Variable "rmcReceived" was initialized to False at the beginning of the If-Else statement. Assign True to the variable to save remote control signal received state.

Select Data > Variable and place the module after the previous modul.

Set Type : bool
Enter Variable Namue : rmcReceived
Assign Constant Value : True
Select Constant Value : True.



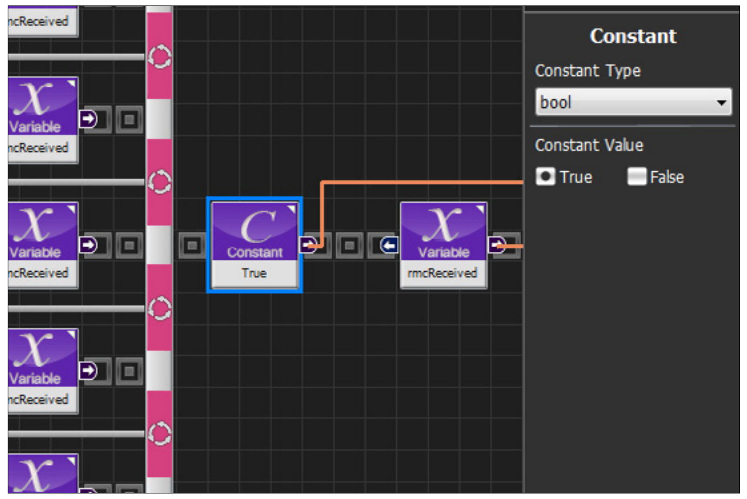
21 Setup Variables “scale” and “rmcReceived” (Mi ~Do)

Apply steps 17~20 to third ~ eighth program line except for the variable “scale” values which need to be changed to values corresponding to 3 Octave “Mi” ~ 4 Ocatve “Do”.

Third program line scale value : 29
Fourth program line scale value :30
Fifth program line scale value :32
Sixth program line scale value:34
Seventh program line scale value:36
Eighth program line scale value:37
Leave last program line blank.

22 Summary

If-Else statement receives input from the remote control and assigns value to variables “scale” and “rmcReceived” according to the input received. When remote control button 1~8 input is received, variable “scale” value is changed to Octave 3 “Do” ~ Octave 4 “Do” accordingly and “rmcReceived” value initialized to False at the beginning of the While statement is changed to True.

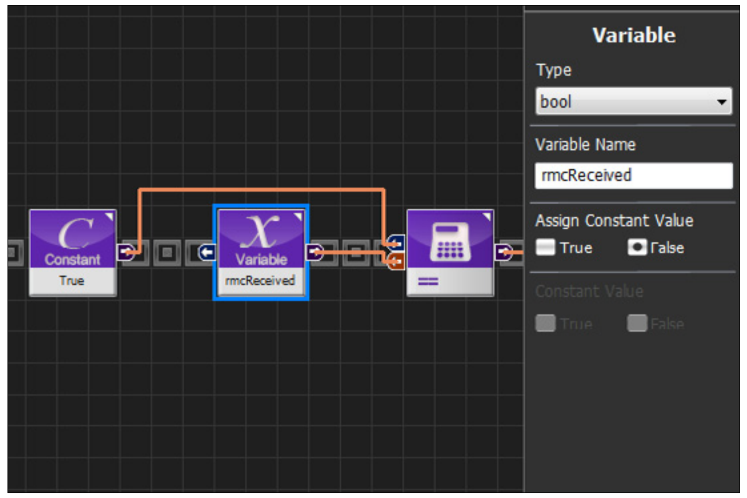

Constant
Constant Type
bool
Constant Value
True False

23 Add Constant

After the If-Else statement, Variable “rmcReceived” value is checked and program enters the section for sounding the buzzer if value is True. Add Constant True.

Select Data > Constant and place the module after the If-Else module.

Set Constant Type : bool.
Set Constant Value: True

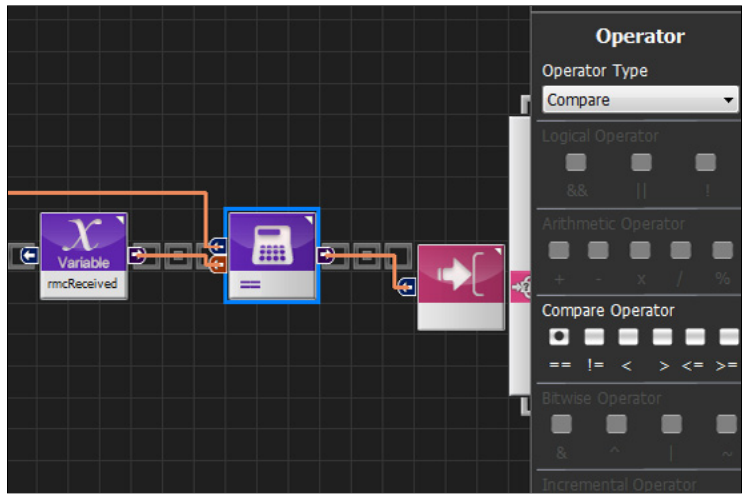

Variable
Type
bool
Variable Name
rmcReceived
Assign Constant Value
True False
Constant Value
True False

24 Add Variable “rmcReceived”

Add rmcReceived to compare the value with True. Since variable value is only read and not changed, Assign Constant Value must be set to False.

Select Data > Variable and place the module after the previous module.

Set Type : bool
Enter Variable Name: rmcReceived
Assign Constant Value : False.

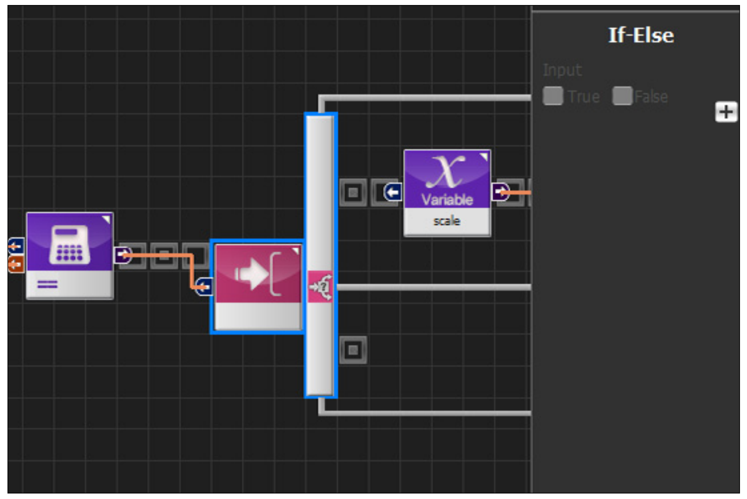

Operator
Operator Type
Compare
Logical Operator
&& || !
Arithmetic Operator
+ - x / %
Compare Operator
== != < > <= >=
Bitwise Operator
Incremental Operator

25 Add Operator Module

Add operator to compare rmcReceived to True.

Select Data > Operator and place the module after the previous module.

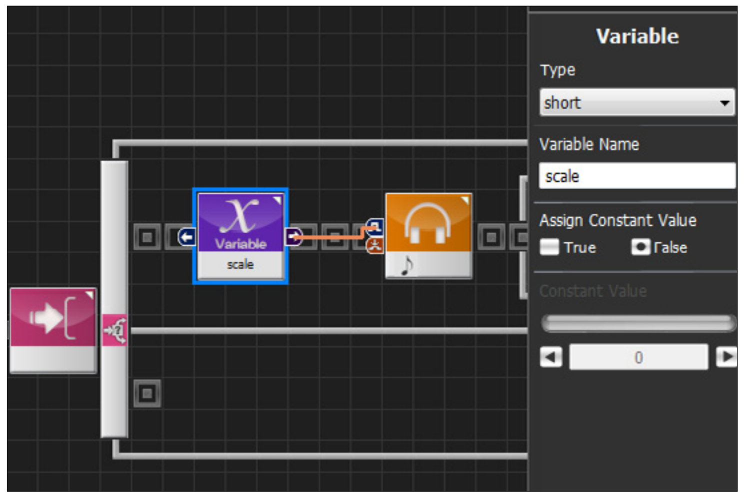
Select Operator Type : Compare
Set Compare Operator : ==
Output True if “rmcReceived” value is equal to True, otherwise output False.


If-Else
Input
True False

26 Add If-Else Statement

Add If-Else statement and connect previous conditional statement to the If-Else statement.

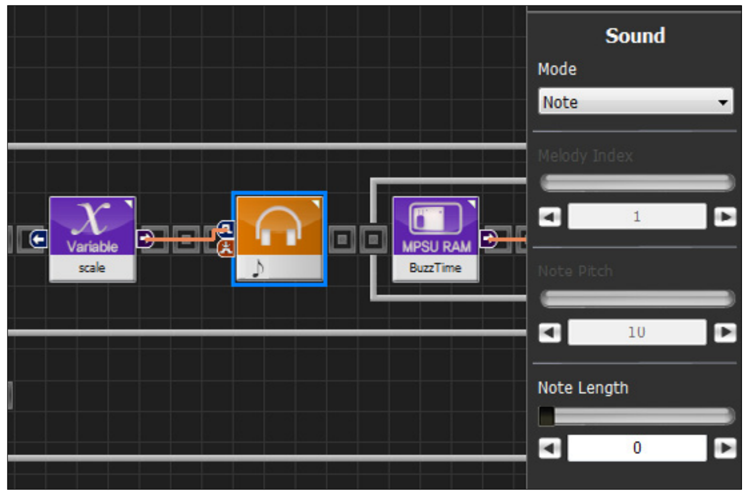
Select Flow > If-Else and place the module after the previous module. Connect output pin from the Operator module from step 25 to the input pin of the If-Else module.


Variable
Type: short
Variable Name: scale
Assign Constant Value: True False
Constant Value: 0

27 Add Variable “scale”

Add variable “scale” which will be used as the input value for the Sound module. Since variable value is only read and not changed, Assign Constant Value must be set to False.

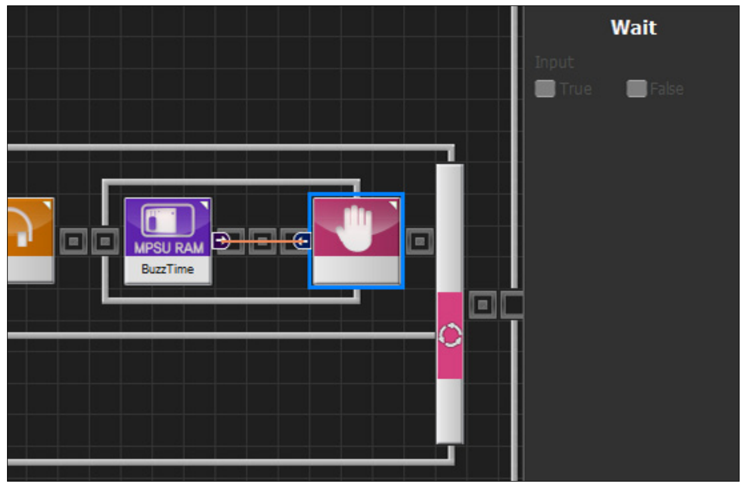
Select Data > Variable and place the module in the first program line of the If-Else module.
Set Type : short
Enter Variable Name: scale
Assign Constant Value : False.


Sound
Mode: Note
Melody Index: 1
Note Pitch: 10
Note Length: 0

28 Add Sound Module

Add sound module with mode set to play single note. Input from the variable “scale” is used as the value of the pitch.

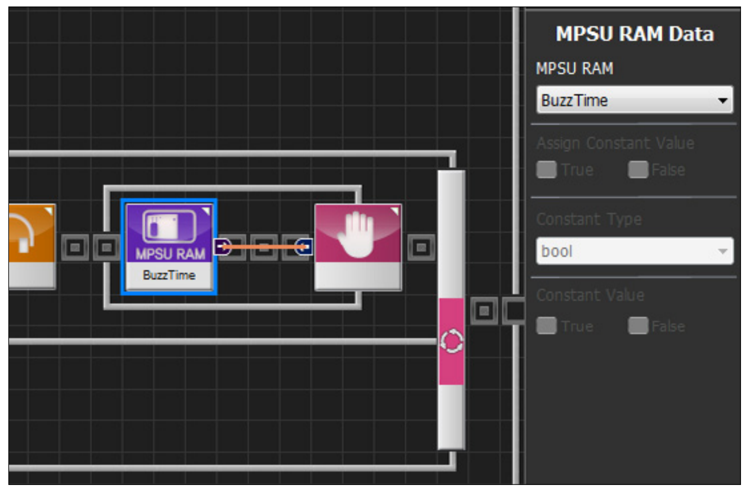
Select Motion > Sound and place the module after the previous module.
Select Mode : Note
Set Note Length : 0.
Connect the output pin from the variable “scale” to the first input pin of the sound module. Note Pitch in the property window become inactive and sound module uses the input value from the “scale” as the Note Pitch value.


Wait
Input: True False

29 Add Wait

Sound module added in the previous step does not wait until the note from the previous command ends before starting new buzzer command. This rapid execution of the while statement will stack the buzzer commands in the internal memory of the DRC-005T and result in buzzer sound lasting much longer than intended. Wait module is added to prevent such a stacking of commands.

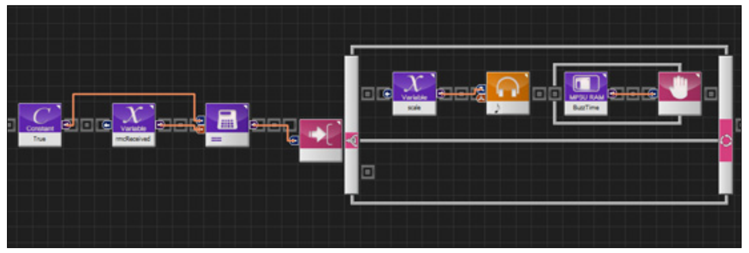
Select Flow > Wait and place the module after the previous module.


MPSU RAM Data
MPSU RAM: BuzzTime
Assign Constant Value: True False
Constant Type: bool
Constant Value: True False

30 Add BuzzTime Module

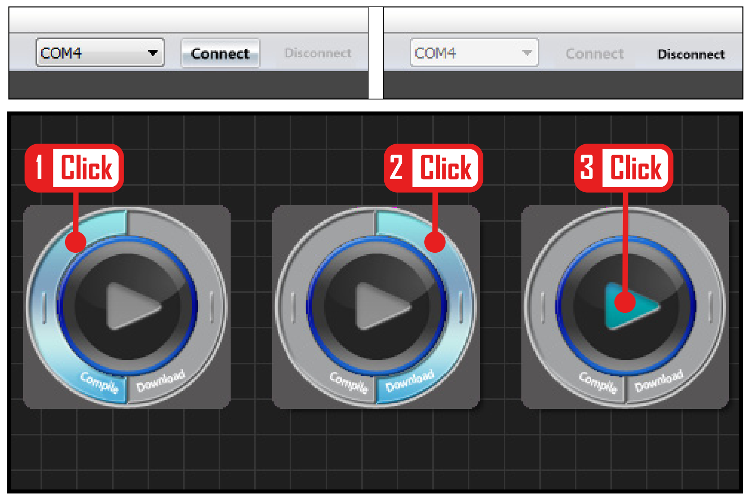
Register BuzzTime in Data > MPSU RAM is used to count down the duration of the buzzer sound. BuzzTime normally maintains value 0 until buzzer starts to sound in which case value changes to the length of the buzzer note. Value decreases as the note is played until it reaches 0 at which time buzzer stops. When BuzzTime module is connected to the input of the Wait module, program will pause at the Wait module until the buzzer note completes playing

Select Data > MPSU RAM and place the module inside the Wait module.
Select MPSU RAM : BuzzTime.
Connect output pin of the BuzzTIme to the input pin of the Wait module.



31 Summary

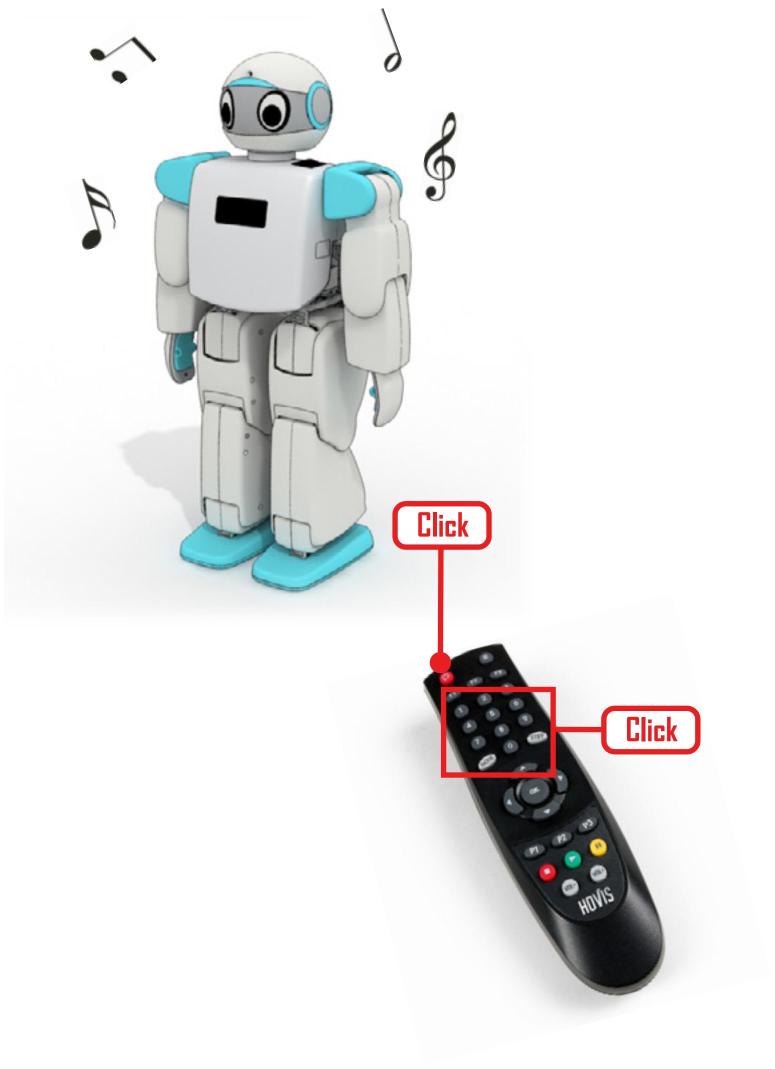
When input from the remote control is detected, variable “rmcReceived” is checked for value “True”. If “rmcReceived” is True, program proceeds to the first line of the If-Else statement where Sound module is run using the “scale” value. Program uses BuzzTime and Wait module to wait until the buzzer note ends and then goes back to the beginning of the while statement.



32 Download

After programming is completed.
Compile -> Download -> Run.

Select the appropriate COMPORT or USB-to-Serial number located on the PC and press Connect to open the serial port.
Click ‘Compile’ and then click ‘Download’ on the right to download the program to the robot if no compile error is found.
Click “Run” button (arrow in the middle) after completing the download.



33 Robot Motion

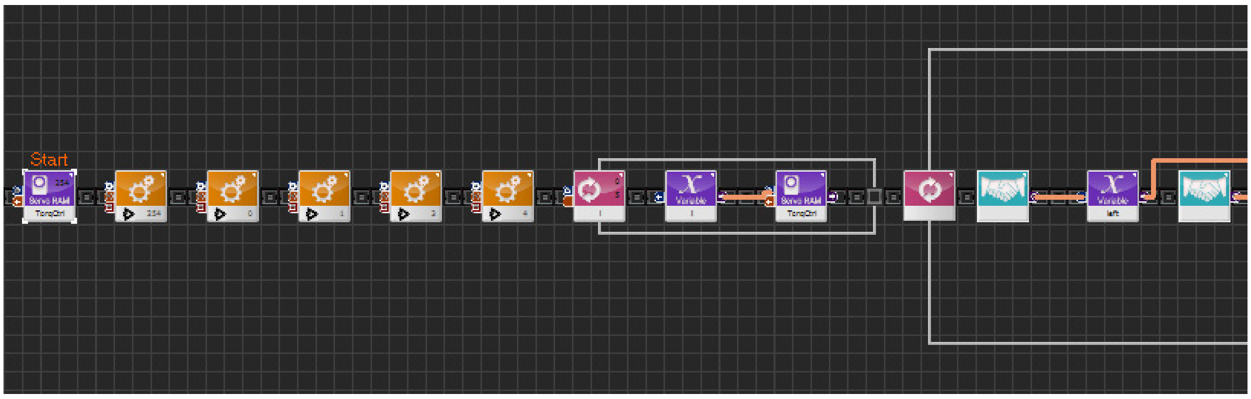
Press remote control buttons 1~8 to play the buzzer notes. Long power button press ends the Task.

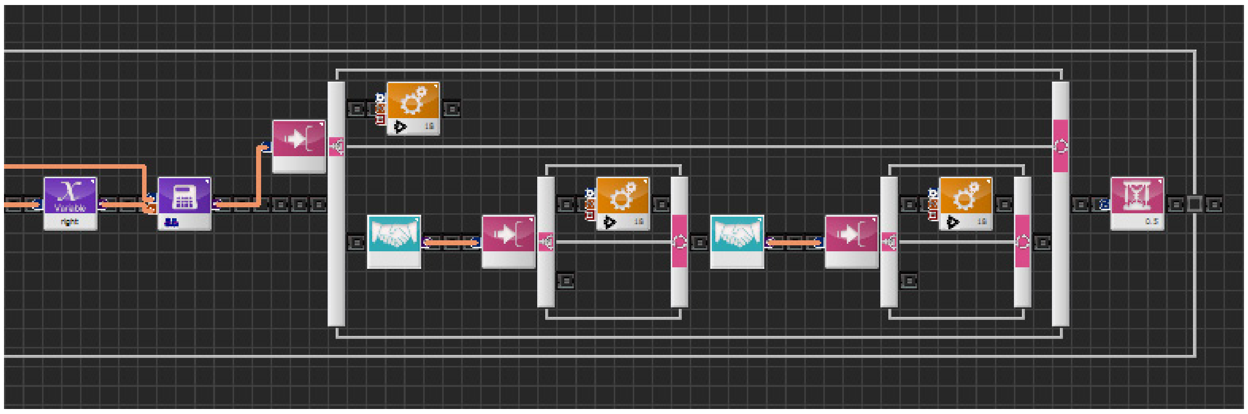
08-11 Programming Individual Module Hand Tact Switch (HOVIS Eco Plus)

Example Description

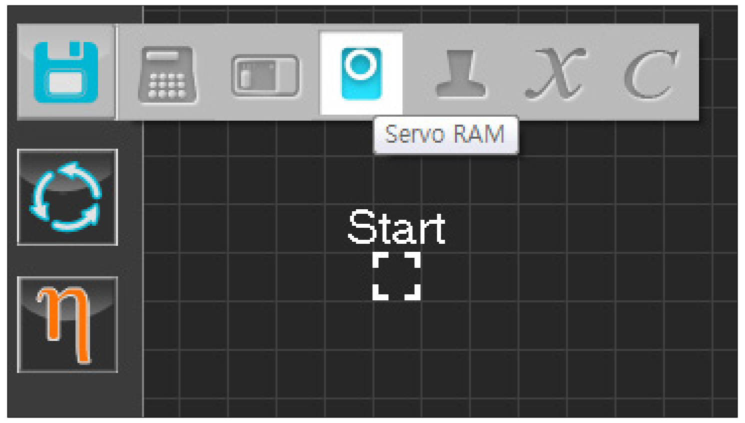
Hovis Eco Plus has touch sensor (Tact Switch) installed to the palm of each hand. This example will use the touch sensors to control the direction of the robot's head. Robot will look to the left when left touch sensor is pressed, look to the right when right sensor is pressed, and look forward when sensors on both hands are touched simultaneously.

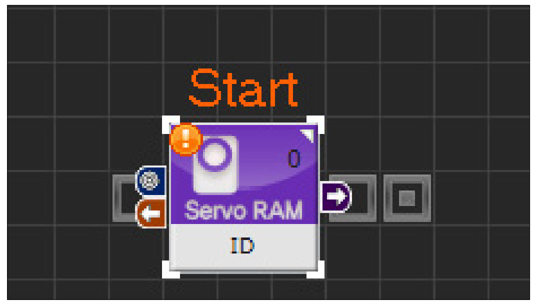
Entire Program and C-Like





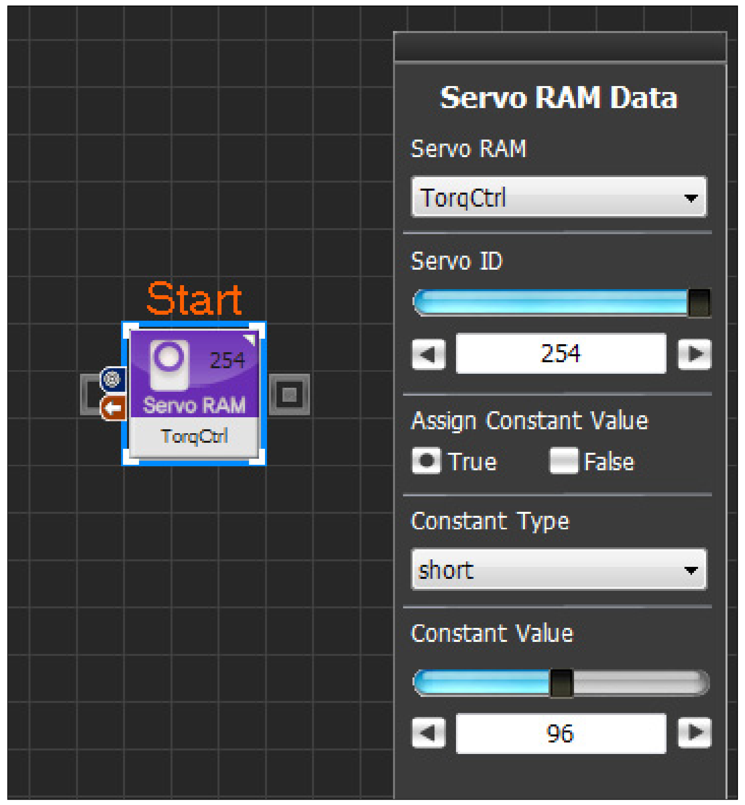
```
short i, left, right
void main()
{
        SERVO_TorqCtrl [254] =96
        jog( 512, 0, 254, 60 )
        jog( 235, 0, 0, 60 )
        jog( 235, 0, 1, 60 )
        jog( 789, 0, 3, 60 )
        jog( 789, 0, 4, 60 )
        for( i = 0 ~ 5 )
        {
                SERVO_TorqCtrl [i] =0
        }
        while( true )
        {
                left=( SERVO_GPIO1 [5] ==0 )
                right=( SERVO_GPIO1 [2] ==0 )
                if( ( left && right ) )
                {
                        jog( 512, 0, 18, 60 )
                }
                else
                {
                        if( ( SERVO_GPIO1 [5] ==0 ) )
                        {
                                jog( 712, 0, 18, 60 )
                        }
                        else
                        {
                        }
                        if( ( SERVO_GPIO1 [2] ==0 ) )
                        {
                                jog( 312, 0, 18, 60 )
                        }
                        else
                        {
                        }
                }
                delay( 500 )
        }
}
```





01 Start programming

Select and drag Servo Ram module to the Start point. Module becomes active when it is properly docked to the Start point.

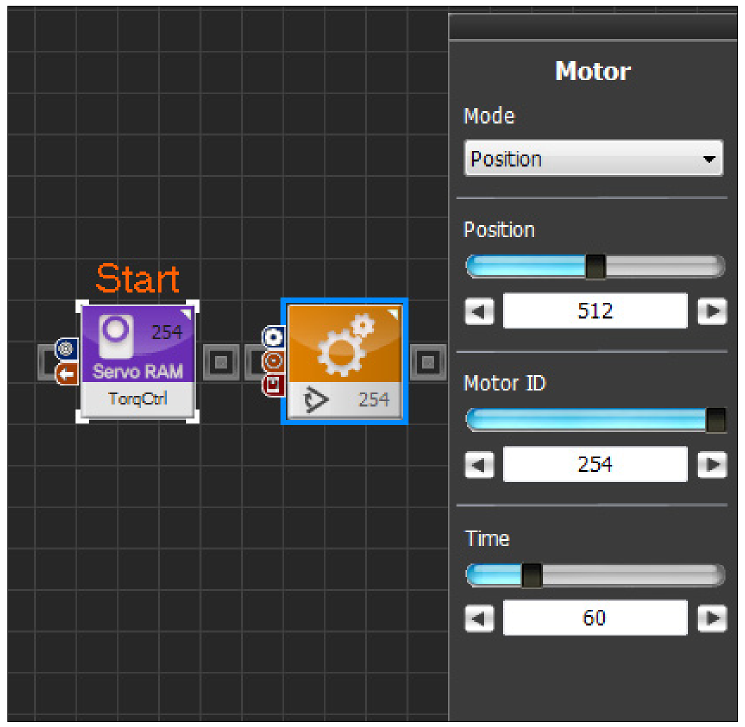


Servo RAM Data
Servo RAM: TorqCtrl
Servo ID: 254
Assign Constant Value: True False
Constant Type: short
Constant Value: 96

02 Turn robot torque ON

Robot operation or motion involves operating the robot servo motors. Servo torque must be turned ON prior to operating the servo. Turn ON the servo torque by assigning following values.

[Servo RAM Module Setup]
Servo ID : 254
Assign Constant Value : True
Constant Type : short
Constant Value : 96

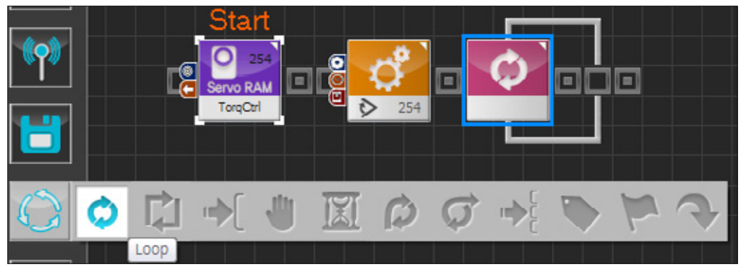


Motor
Mode: Position
Position: 512
Motor ID: 254
Time: 60

03 Basic robot posture

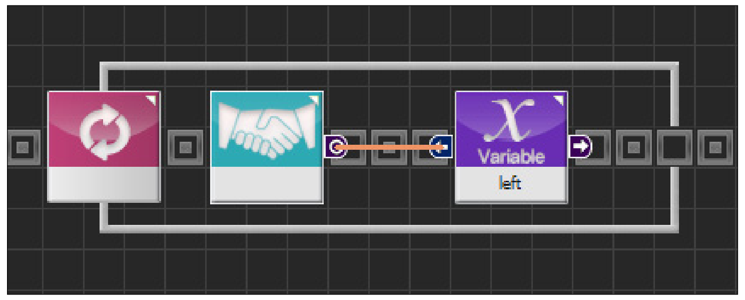
Operate servo motors to set robot posture to basic posture. Setup the motor module as follows.

[Motor Module Setup]
Mode : Position
Position : 512
Motor ID : 254
Time : 60



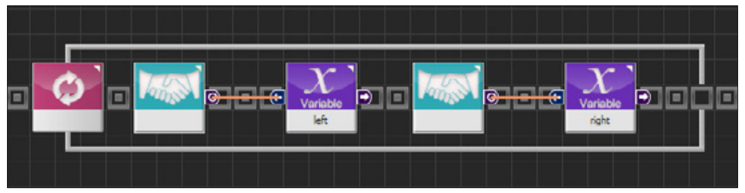
04 Infinite Loop

Place loop module in the program to continuously detect touch to the sensors



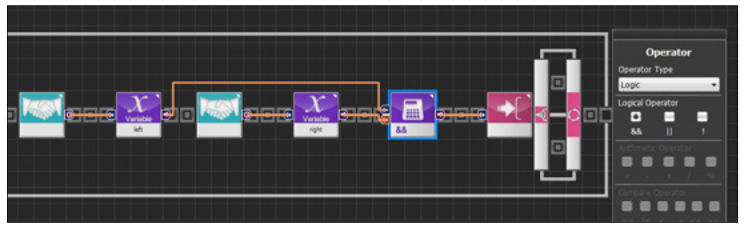
05 Left hand touch detection variable setup

Place Hand Touch Sensor module and the Variable module as shown in the diagram. Connect the output pin from the Hand Touch Sensor module to the input pin of the Variable module. Click Hand Touch Sensor module and assign Hand as Left. Click Variable module and assign name as 'left'.



06 Right hand touch detection variable setup

Place Hand Touch Sensor module and Variable module as shown in step 05. Assign Hand as Right and variable name as 'right'.

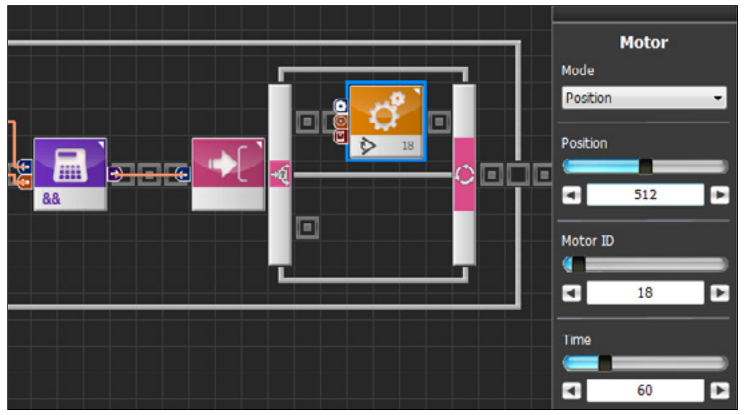


07 Detecting and deciding touch to both hands

Place Operator and If-Else modules in sequence as shown in the diagram. Select and setup the Operator module as follows.

[Operator module setup]
Operator Type: Logic
Logical Operator: &&

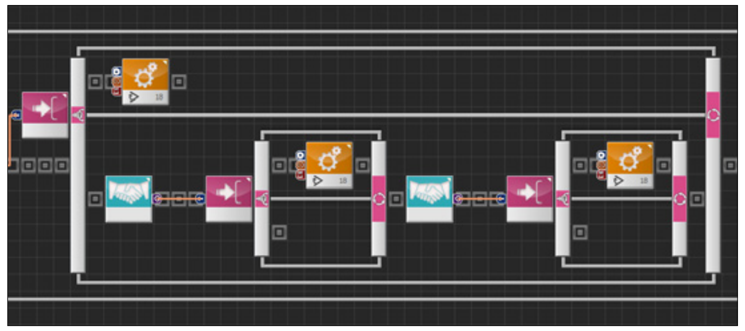
Connect output pins from the 'left' and 'right' variable modules to the input pin of the Operator module. Connect output pin from the Operator module to the input pin of the If-Else module.
When both hands were touched simultaneously, program will process the True section of the If-Else module and process False otherwise.



08 Both hands touched simultaneously

When both hands were Touched simultaneously, program controls the servo motor to turn the robot head to look forward direction.

[Motor module setup]
Mode: Position
Position: 512
Motor ID: 18
Time: 60



[Left hand setting]
Hand Touch Sensor Module
Hand : Left

[Motor Module]
Mode : Position
Position : 712
Motor ID : 18
Time : 60

[Right hand setting]
Hand Touch Sensor Module
Hand : Right

[Motor Module]
Mode : Position
Position : 312
Motor ID : 18
Time: 60

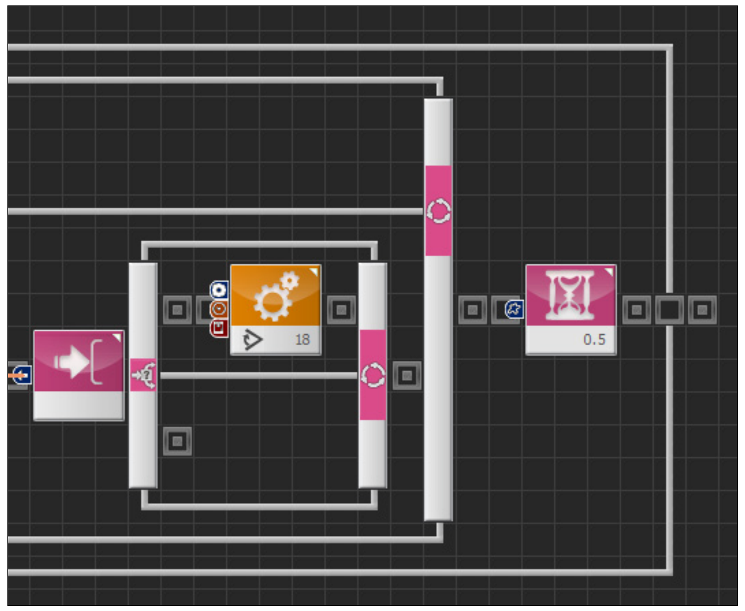
09 Both hands not touched simultaneously

When both hands were not touched simultaneously, program checks for touch to either the left or the right hand and controls the servo motor to turn the head towards the direction of the touched hand.
Left hand check setup

[Hand Touch Sensor Module]
Hand : Left
[Motor module]
Mode : Position
Position : 712
Motor ID : 18
Time : 60

Right hand check setup

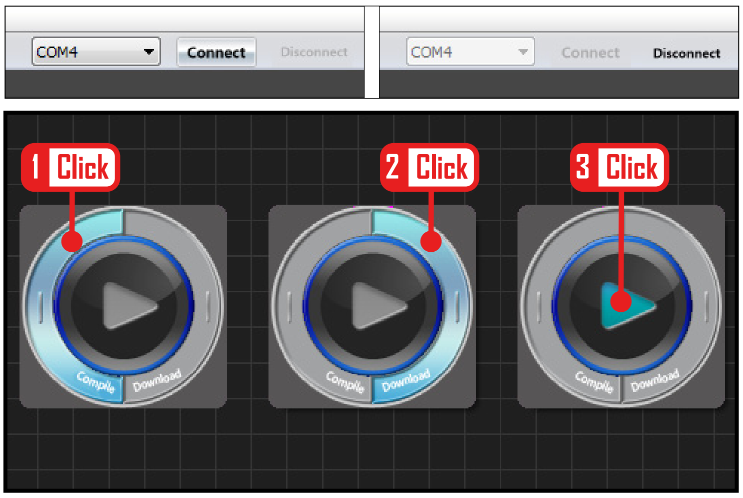
[Hand Touch Sensor Module]
Hand: Right
[Motor module]
Mode : Position
Position : 312
Motor ID : 18
Time : 60



10 Touch sensor detection interval control

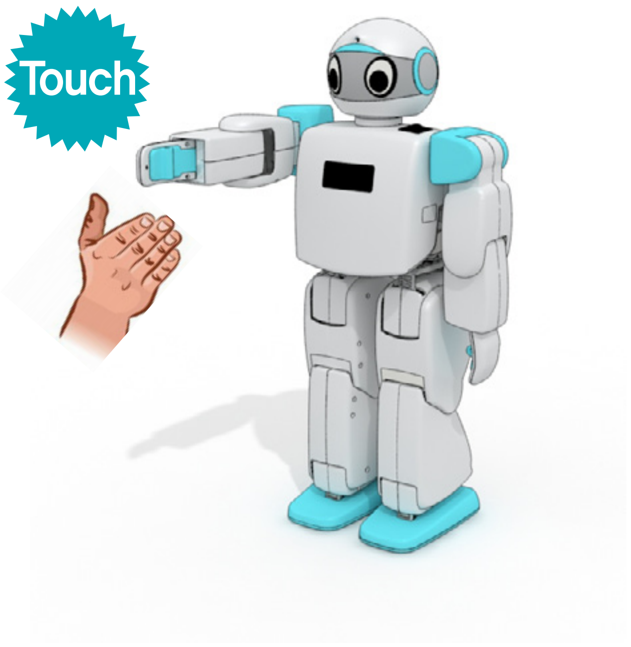
Add delay module to control the touch sensor detection time interval. Shorter delay time increases the sensitivity and vice versa.

[Delay module setup]
Time: 0.5
Time: 0.5



11 Compile, download, run

Click left compile button to compile the program. If there is no error, click right download button to download the program to the robot. Click run button to run the program.



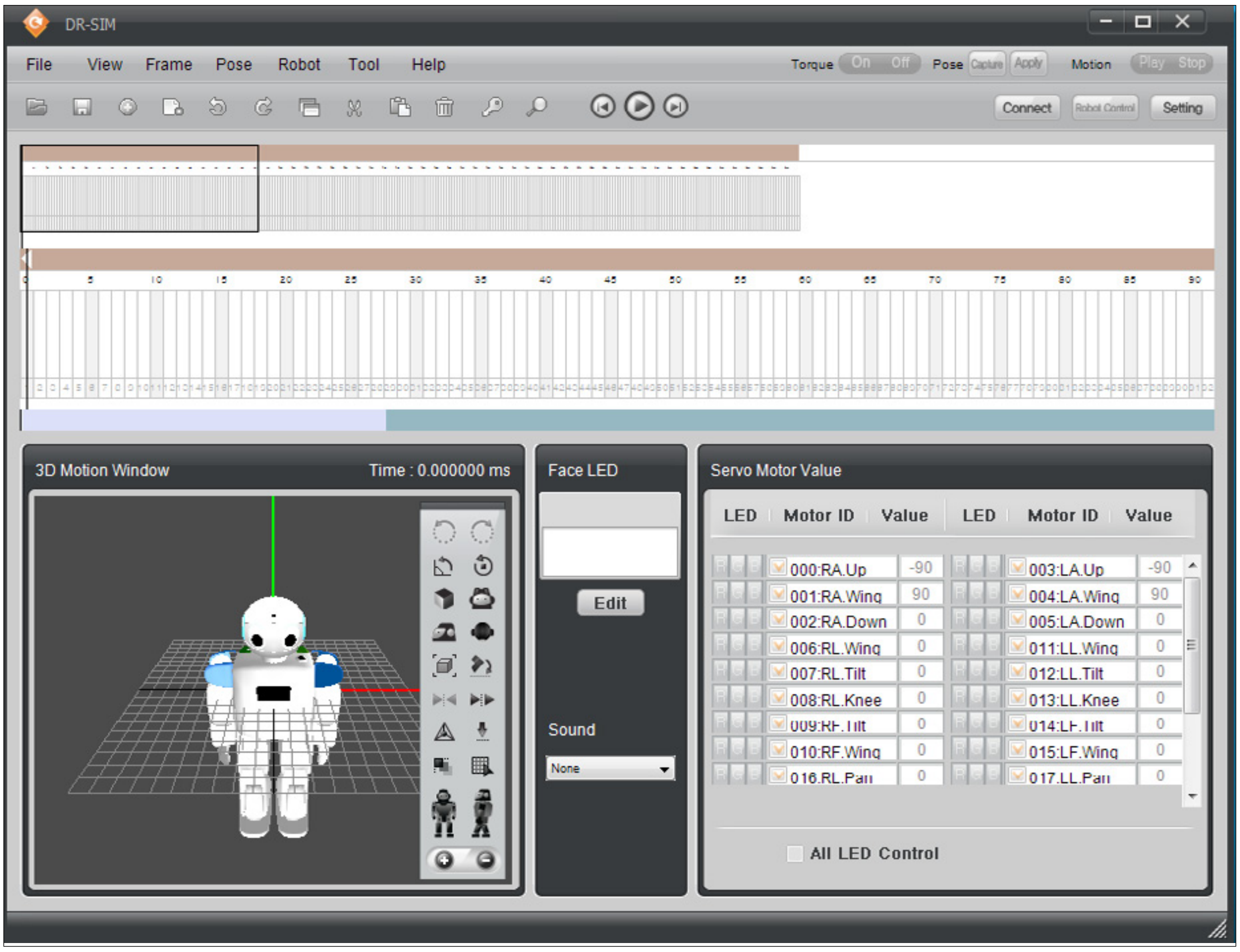
12 Robot motion

Touch left and right hand to operate the robot.

Programming Individual Module
Head Touch Sensor, LED (HOVIS Eco Plus)

Editing and Downloading head LED

Controller contains predefined face LED effects applicable to each emotion and motion. These standard face LED effects can be edited using DR-SIM and downloaded to the controller. Refer to the PART 6 Software: DR-SIM Multimedia Effect for more detailed information.

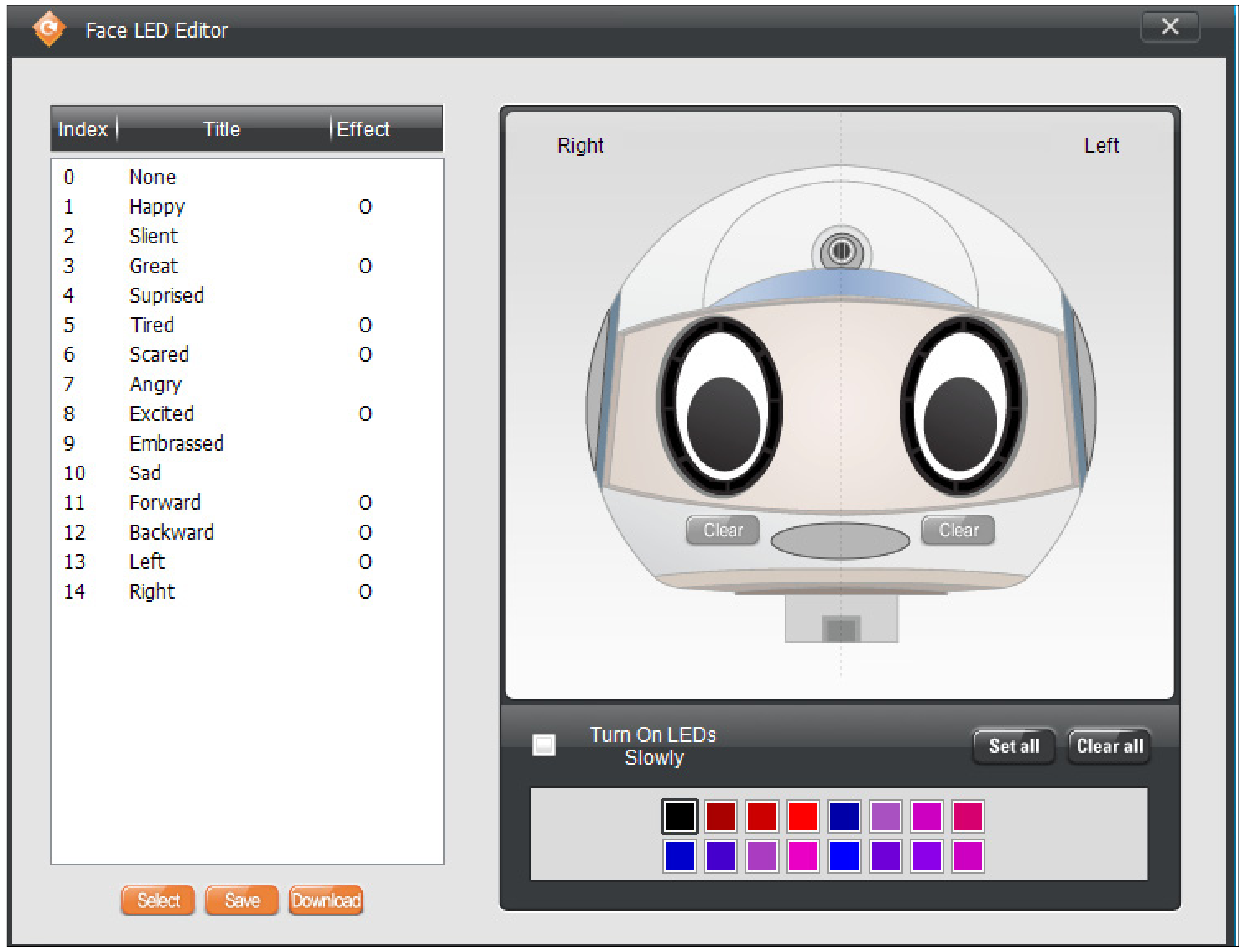


Time : 0.000000 ms

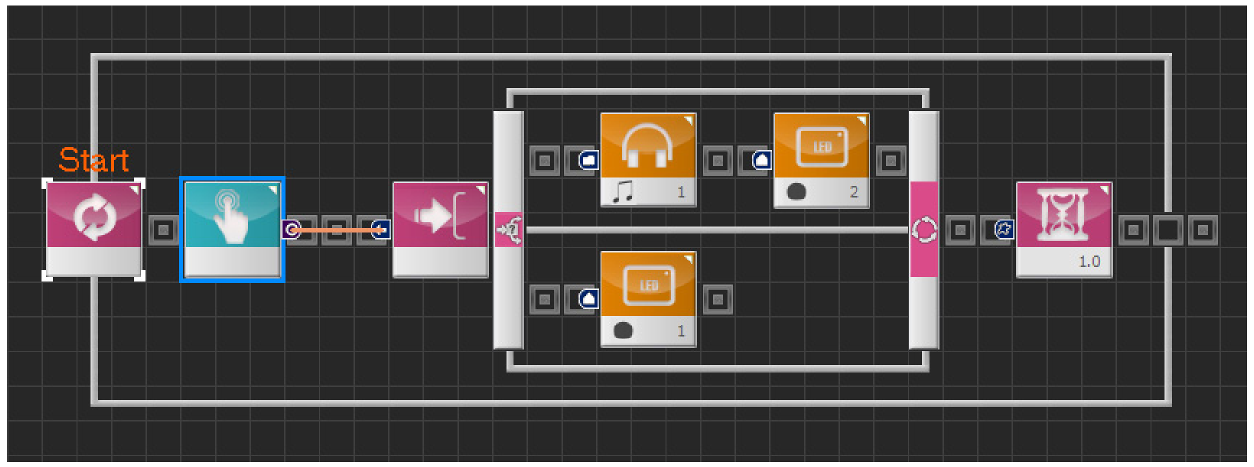
LED	Motor ID	Value	LED	Motor ID	Value
	000:RA.Up	-90		003:LA.Up	-90
	001:RA.Wing	90		004:LA.Wing	90
	002:RA.Down	0		005:LA.Down	0
	006:RL.Wing	0		011:LL.Wing	0
	007:RL.Tilt	0		012:LL.Tilt	0
	008:RL.Knee	0		013:LL.Knee	0
	009:RF.Tilt	0		014:LF.Tilt	0
	010:RF.Wing	0		015:LF.Wing	0
	016:RL.Pan	0		017:LL.Pan	0

Example Description

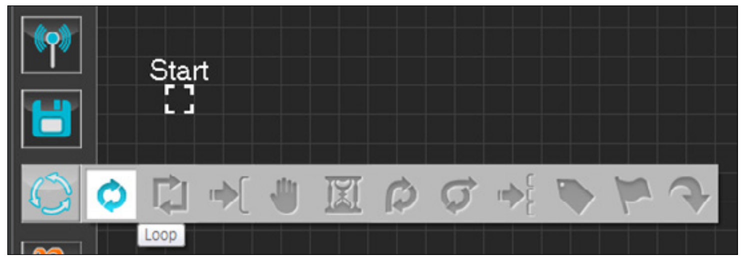
This example will use the head touch to control the face LED effect. Robot will express silent(Index:2) face LED effect with sound when head is touched and express happy(index:1) face effect when touch is released.


Face LED Editor

Index	Title	Effect
0	None	
1	Happy	O
2	Slient	
3	Great	O
4	Suprised	
5	Tired	O
6	Scared	O
7	Angry	
8	Excited	O
9	Embrassed	
10	Sad	
11	Forward	O
12	Backward	O
13	Left	O
14	Right	O

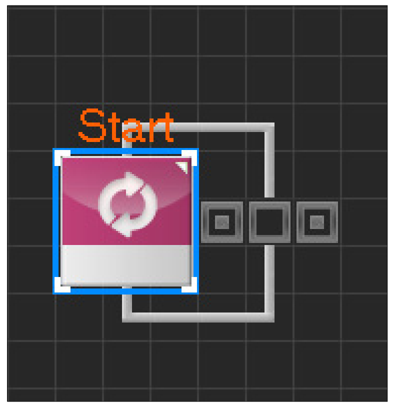


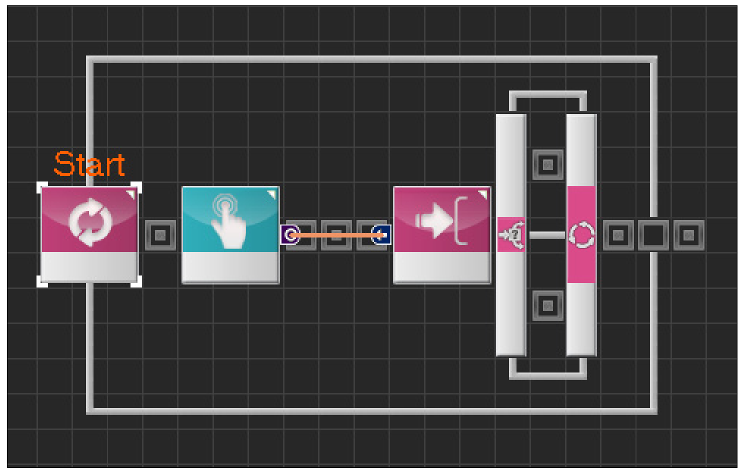
```
void main()
{
          while( true )
          {
                    if( ( MPSU_TouchStatus ) )
                    {
                              melody( 1 )
                              headled( 2 )
                    }
                    else
                    {
                              headled( 1 )
                    }
                    delay( 1000 )
          }
}
```



0 1 Start Programming

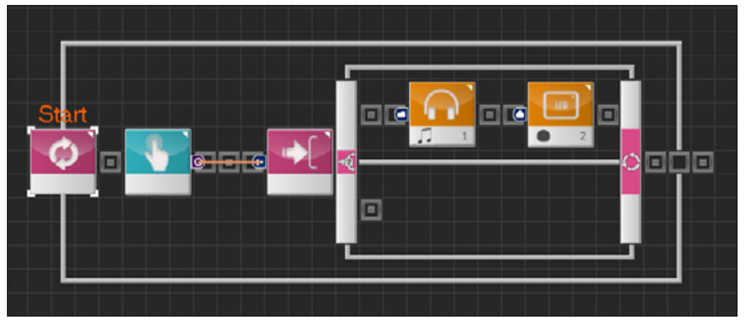
Select and drag Loop module to the Start point. Loop module becomes active when it is properly docked to the Start point.





02 Head touch decision

Place Touch Sensor module and If-Else module as shown in the diagram. Connect output pin of the Touch Sensor module to the input pin of the If-Else module.
Touch Sensor module will output True when robot head is touched and output False otherwise.



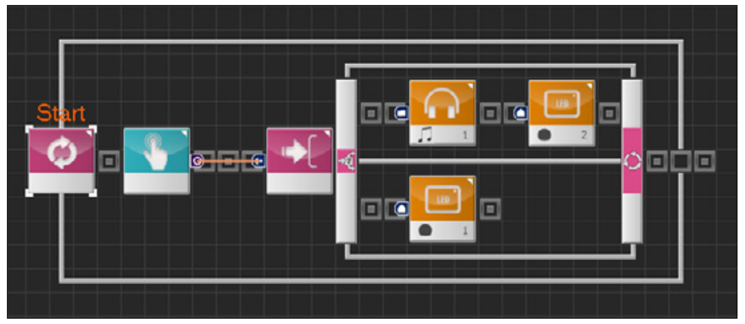
03 Head Touched

Robot will output melody and express silent face LED effect when head is touched.
Place Sound and LED module as shown in the diagram and assign following values.

[Sound module setup]
Mode : Melody
Melody Index: 1

[LED module setup]
Mode : Head LED
LED Index : 2 (Silence)

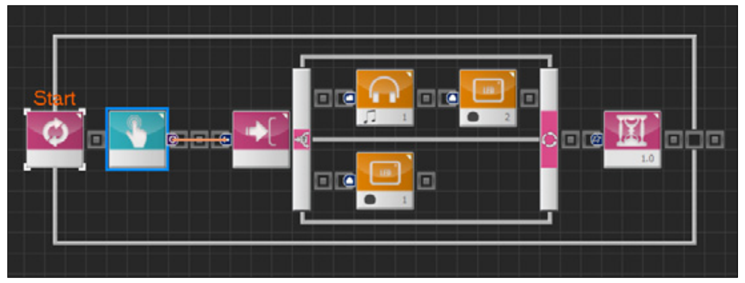
※ Refer to example description for LED index.



04 Head not touched

Express happy LED face effect when head is not touched.
Place LED module as shown and assign LED Index value.

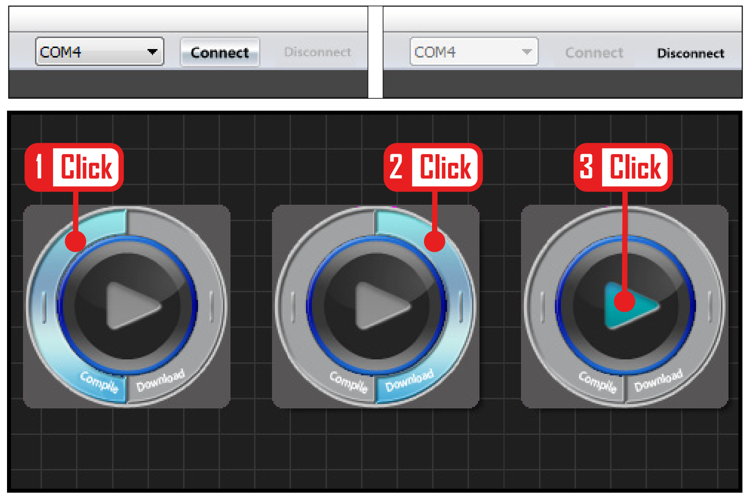
[LED module setup]
Mode : Head LED
LED Index : 1 (happy)
※ Refer to example description for LED Index



05 Touch sensor detection interval control

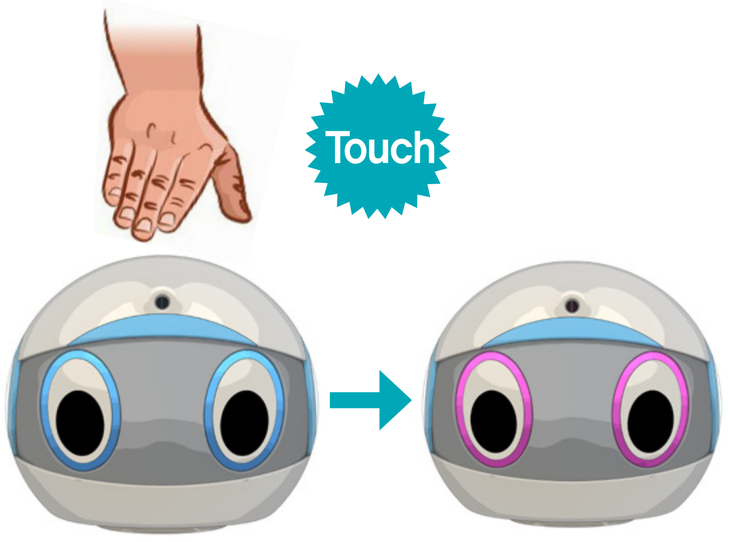
Add delay module to control the touch sensor detection time interval. Shorter delay time increases the sensitivity and vice versa.

[Delay module setup]
Time : 0.5



06 Compile, download, run

Click left compile button to compile the program. If there is no error, click right download button to download the program to the robot. Click run button to run the program.



07 Robot motion

Touch the robot head to view the head LED effect.