Robo-128
Wiring I/O mobile robot

the autonomous robot
programming with C/C++

Activity manual
version 1.0

INNOVATIVE EXPERIMENT

Credits

Contents

Chapter 1
Robo-128 Introduction

1.1 Robo-128 part list

1. RBX-128 controller board with on-board digital compass sensor
2. mini-B USB cable
3. ZX-InfraredEYE : Multi-direction infrared sensor
4. ZX-SWITCH : Switch/Touch sensor x 2
5. BO2-48:1 DC motor gearbox with holder and cable x 3
6. RC servo motor standard type
7. Omni-directional wheel with metal coupling for BO2 gearbox x 3
8. 3AA batter holder with cable x 2
9. Plastic joiner and Strip joiner set
10. Right angle metal part set
11. Nuts and screws pack
12. CD-ROM contains software, example code and doucmentation
13. Printed documentation

1.2 RBX-128 Controller board features

Figure 1-1 shows the main components of RBX-128 controller board of Robo-128. It includes :

l Main microcontroller is Atmel's ATmega128. It features 8-ch 10-bit Analog to Digial Converter, 128-KByte Flash memory , 4-KByte EEPROM, 4-KByte RAM. Operated with 16MHz clock from external crystal

l Define all ports compatible with Wiring I/O standard hardware (www.wiring.org.co). The number of port are 0 to 50.

l 11 programmable port in JST connector type. Includes A/D port (7 : port 40/ADC0 to port 46/ADC6), Two-wire interface or TWI (2 : port 0/SDA and port 1/SCL) and UART serial port communication (2 : port 2/RX1 and port 3/TX1). Both TWI and UART ports can config to digital input/output port for more I/O applications.

l Analog input (ADC0 to ADC6) supports 0 to +5Vdc input. The converter resolution is 10-bit. The result value is 0 to 1,023 range.

l One variable resistor is connected with the Analog input ADC7 of main microcontroller for simple ADC experiment.

l 2 of button switches with resistor pull-up are connected with port 49 and 50 of the Wiring I/O controller board for simple digital input experiment.

l One LED with a current limited resistor. It is connected with port 48

l One piezo speaker at port 4

l 16x2 characters LCD for monitoring

l On-board digital compass; HMC6352 from Honeywell. It is interfaced by I2C bus or TWI

l UART port for interfacing serial module device such as camera module (ZX-CCD, CMUCAM1, CMUCAM2, mCAM), servo controller board (Parallax servo controller, ZX-SERVO16U), Real-time clock (ZX-17 : serial real-time clock moduel)

l 6-ch DC motor driver with indicators. Support 4.5V to 9V DC motor. Maximum current output 3A and 1.2A continuous.

l 8-RC servo motor output; support 4.8 to 7.2V standard and continuous servo motor types.

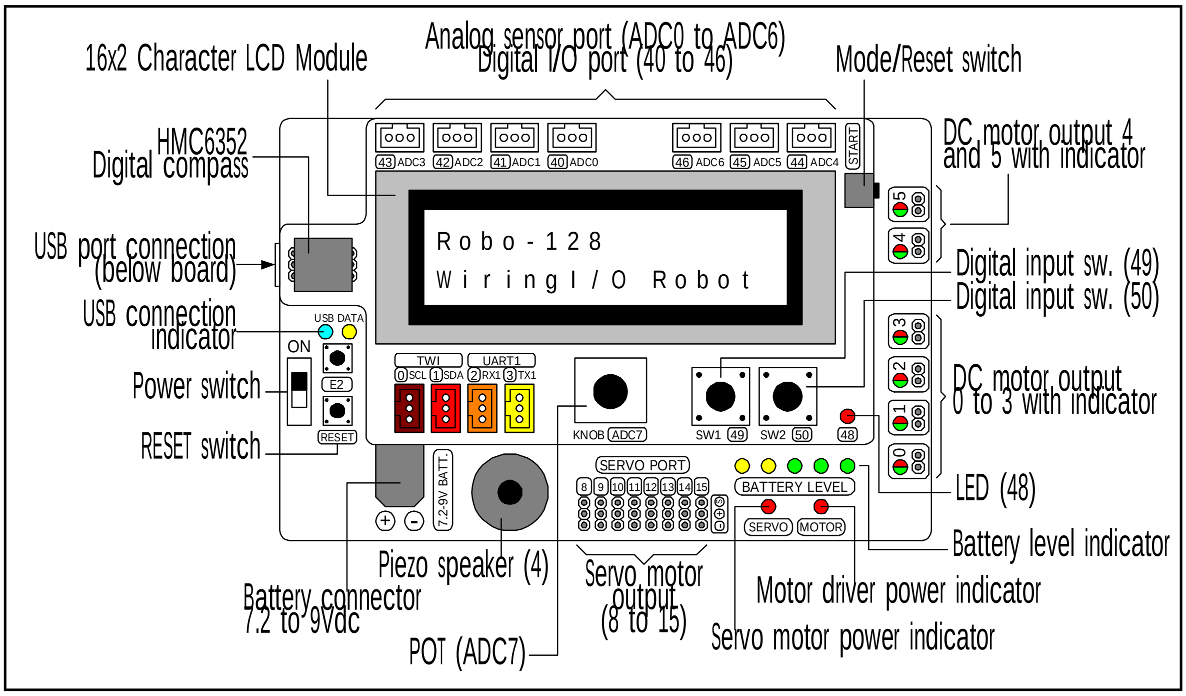


Figure 1-1 : Main controller board of Robo-128 layout

l Motor driver power indicator; nomally turned on. It will off when motor is short-circuit.

l 5-status battery level monitor circuit :

- Last left yellow LED displays that the input supply voltage is 6.75V. When the battery voltage is lower than 6.75V, this LED will start to blink.

- Next yellow LED displays the input supply voltage at 7.0V.

- First green LED displays the input supply voltage at 7.25V.

- Second green LED displays the input supply voltage 7.5V.

- Last right green LED displays that the input supply voltage is higher than at 7.75V.

l Download and interface with computer via USB port by using USB to UART converter chip; FT232RL. USB connection indicator is available.

l Set the operation by Mode/Reset switch

l Supply voltage range +7.2 to +9V 2400mA for 4 motor loads.

l 2 voltage regulator on-board; +5Vdc for microcontroller and all digital circuit, +6Vdc for all motor driver circuits. By using voltage regulator; the motor driver circuit can drives DC motor with constant speed when battery voltage is full and reduce to 60%. It features constant speed without effect by battery voltage until it is lower 60% of full.

1.3 Output device features

1.3.1 DC motor gearbox

DC motor gearbox of Robo-128 is the BO2 model. The technical features are as follows :

l Requires the supply voltage +4.8 to +9Vdc 130mA @6V and no load

l Gear ratio 48:1

l Speed 250 round per minute @6V and no load

l Weight 30 gram

l Torque 0.5kg.-cm.

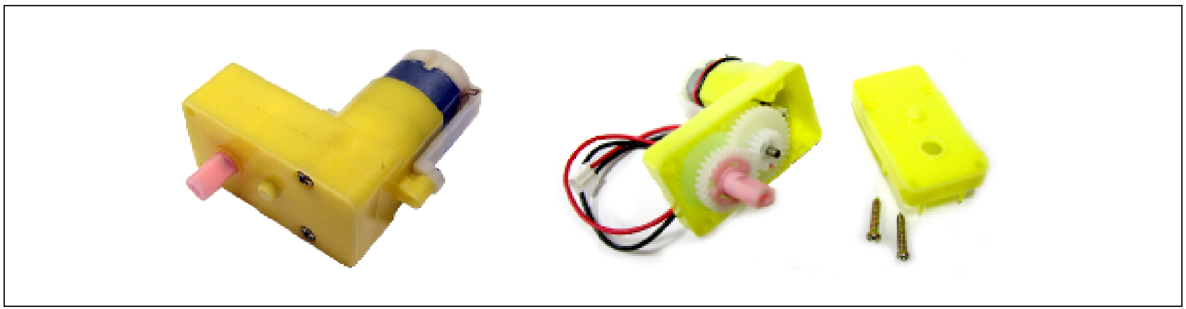


Figure 1-2 : BO2-48:1 Dc motor gearbox of Robo-128

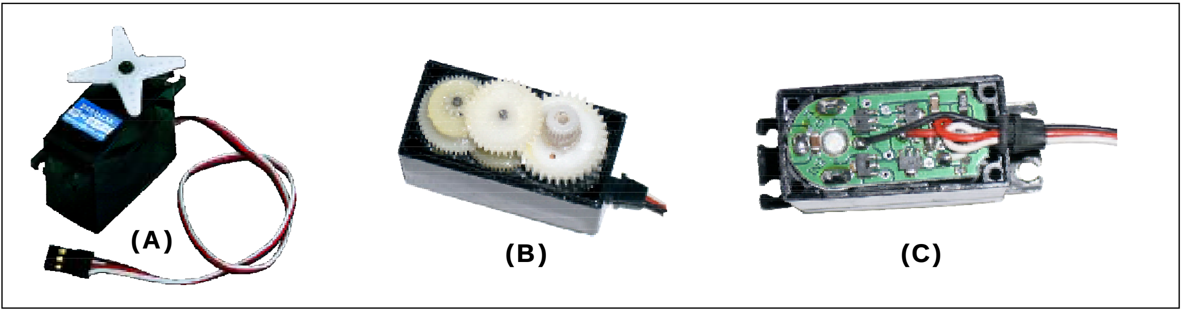


Figure 1-3 : Details of standard RC servo motor of Robo-128
(A) outside body (B) Gear system (C) Electronic circuit controller board

1.3.2 Standard servo motor

Servo motor has 3 wires; Signal (S), Supply voltage(+V) and Ground (G) The technical features is as follows :

- Requires the supply voltage +4.8 to +6Vdc
- Weight 45 gram
- Torque 3.40kg-cm. or 47 oz-inches.
- Size (width x length x height) 40.5 x 20 x 38 mm. or 1.60 x 0.79 x 1.50 inches.

1.4 Robo-128 sensor features

1.4.1 ZX-irEYE (Infrared Eye) Multi-direction infrared detector

Figure 1-4 shows the ZX-irEYE layout. It is used for infrared light detection. It has 2 outputs :

1. Range : It gives 0 to +5V forward the infrared light density detection. The output goes high level when this sensor is near infrared light source and decrease when sensor is far from infrared source.

2. Direction : It gives 0 to +5V direct variation the detected angle of infrared light 0 to 180 degree.

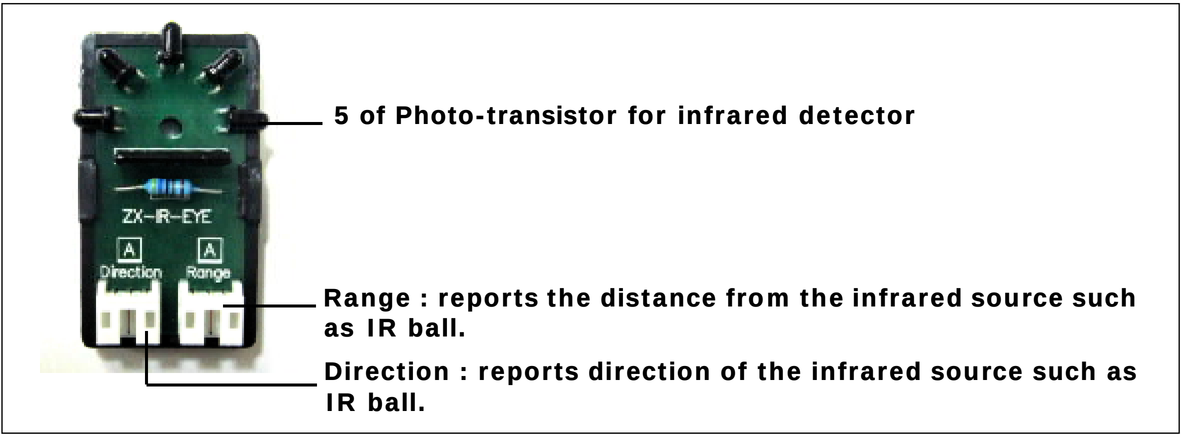

ZX-IR-EYE
Direction
Range
5 of Photo-transistor for infrared detector
Range : reports the distance from the infrared source such as IR ball.
Direction : reports direction of the infrared source such as IR ball.

Figure 1-4 : The layout of ZX-irEYE Multi-direction infrared detector

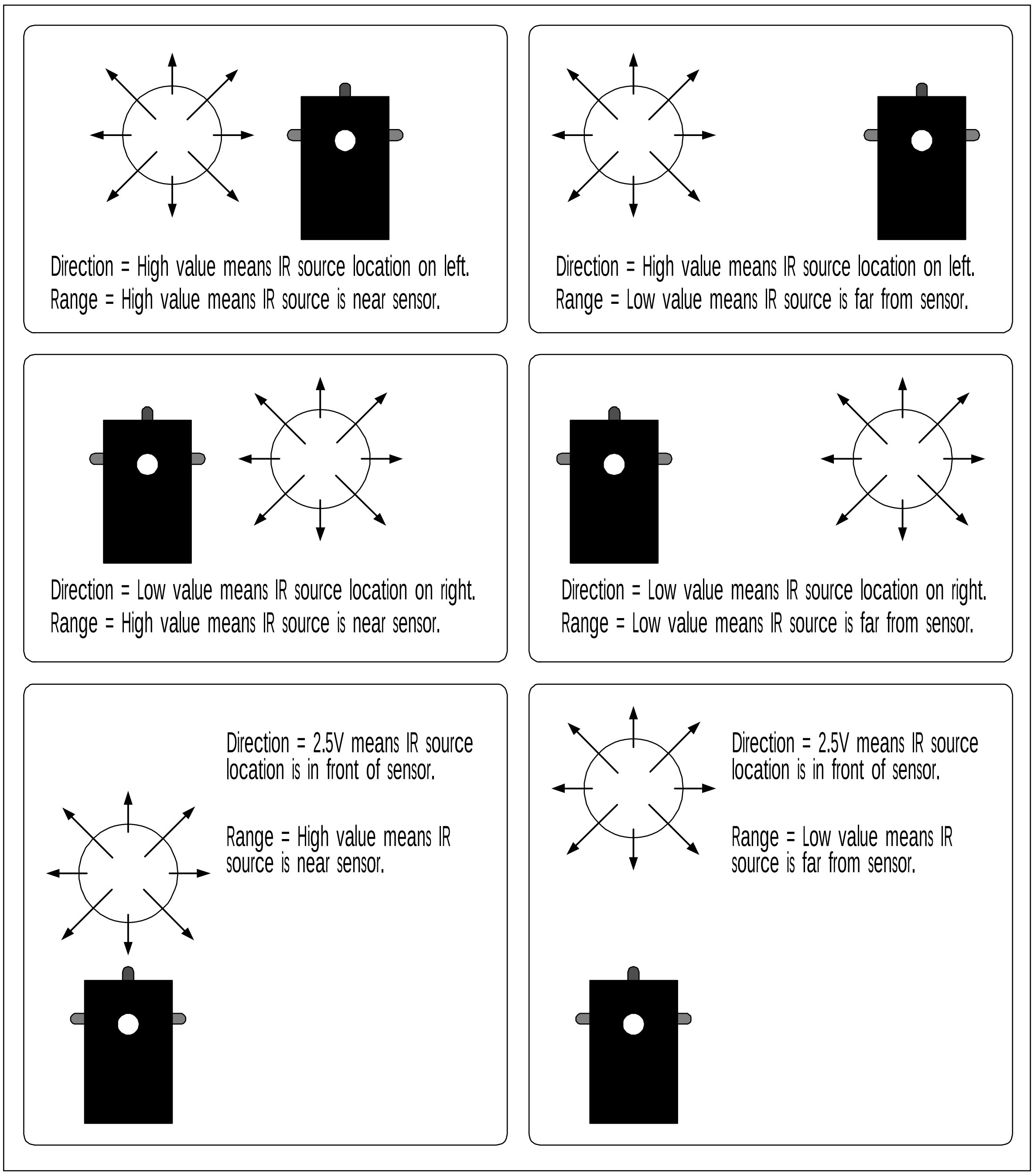

Direction = High value means IR source location on left.
Range = High value means IR source is near sensor.
Direction = High value means IR source location on left.
Range = Low value means IR source is far from sensor.
Direction = Low value means IR source location on right.
Range = High value means IR source is near sensor.
Direction = Low value means IR source location on right.
Range = Low value means IR source is far from sensor.
Direction = 2.5V means IR source location is in front of sensor.
Range = High value means IR source is near sensor.
Direction = 2.5V means IR source location is in front of sensor.
Range = Low value means IR source is far from sensor.

Figure 1-5 : ZX-irEYE operation

The figure 1-5 shows ZX-irEYE in operation. User can interface this sensor with analog input of Robo-128 (ADC0 to ADC6). This sensor is suitable for detecting n the infrared ball (in continuous mode) of the soccer robot games.

1.4.2 HMC6352 digital compass

Honeywell HMC6352 Compass Module is fully integrated and combines 2-axis magneto-resistive sensors with the required analog, digital, microprocessor and algorithms required for heading computation. The HMC6352 Compass Module provides a direct heading angle and includes internal calibration algorithms within its firmware. Temperature compensation and calibration are built in, as well as protection against stray magnetic fields.

Features

- Wide voltage supply range (2.7 to 5.0 Vdc) for use with most microcontrollers
- Digital output (I^2C) to nearest 0.1°
- Built-in calibration for an average error of 2.5°
- Communication: I^2C; 100 kbps max.
- Internally calculates heading, simplifying software demands
- Stray magnetic field protection
- Low current consumption (typically 1µA in standby mode, 1 mA full operation at 3Vdc)
- Compact, breadboard-friendly 6-pin DIP module package

The layout and pin assignment of HMC6352 are shown in the figure 1-11. This module is installed on the controller board of Robo-128. The north pole orientation is shown in the figure 1-11. However user can set the your own reference direction by software. We provides the software library for this purpose already.

Warning ! Do not install this sensor nearby the high magnetic field density area and components such as motor, permanent magnet bar, electromagnetic generator devices. It causes this sensor malfunction and may problaby damaged it if the magentic field density is over 10,000 Gauss.

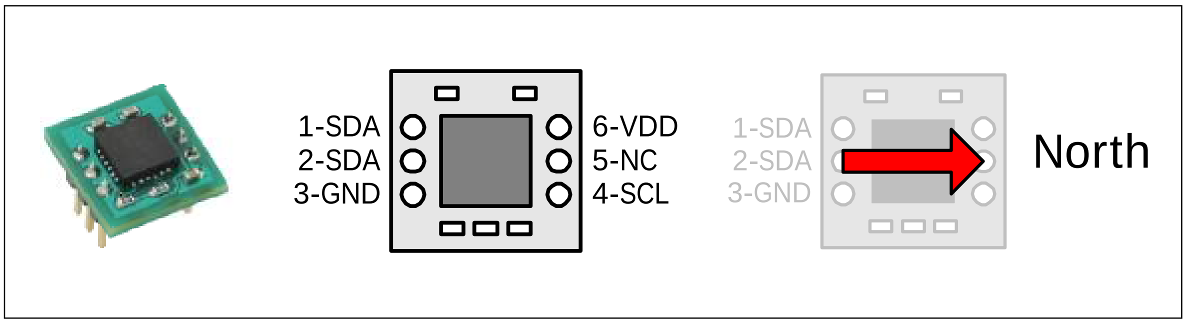
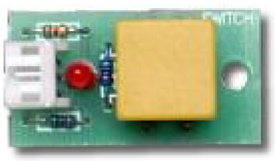
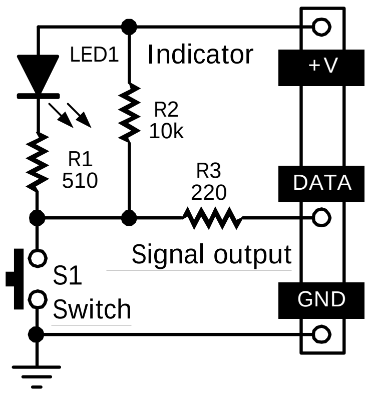


Figure 1-11 Layout, pin assignment and north pole orientation of HMC6352 digital compass module

1.4.3 Switch module/Touch sensor





The switch input is used to detect collision at logic "0". Two sets along with the connecting cable are provided.

1.4.4 Infrared Ball (RCJ-05)

This is an Infrared ball that compliant to the new specification of RoboCupJunior soccer completition.

The 40kHz carrier output of the ball is modulated with a trapezoidal (stepped) waveform of frequency 1.2kHz. This enables the sensors to easily distinguish the infrared light emitted by the ball from infrared emissions from other sources, and also to estimate the approximate distance to the ball.

The new ball is not only able to operate in this new modulated pulse mode but also in the current unmodulated mode, as well as two other pulse modulation modes for other robot competitions. Total of 4 modes available :

A : RCJ-PULSE Pulse modulated RoboCupJunior Soccer Ball Standard (step waveform modulation, 40kHz carrier)

B: RCJ-DC Unmodulated RoboCupJunior Soccer Ball Standard (steady IR emission)

C : 600Hz pulse modulation (40kHz carrier)

D : 1200Hz pulse modulation (40kHz carrier)

For the power indicator, two high-visibility red LEDs are used. They also indicate the battery life at 3 levels by blinking at different speeds.

Technical features of RCJ-05 is as follows :

IR peak wavelength : 940nm.

Number of infrered LED : 20 pieces.

Battery : 4 of AAA size batterry. The alkaline dry cell (AM4/LR03, 1.5V) or Ni-MH rechargeable (1.2V) are recommendded.

Supply voltage (rating) : 4.8 to 6Vdc.

Current consumption : 80mA (mode-A), 230mA (mode-B), 130mA (mode-C, D)

Dimension : 74mm. (3 inches) in diameter.

Weight : 95 g (0.21 lbs) approximation excluding batteries.

For working with the Robo-128, must set the RCJ-05 mode to B. Please see the setting instruction with RCJ-05 documentation inside its box.

1.5 Mechnical part features

1.5.1 Omni-directional wheel

This is special wheel. The commercial name is called Transwheel. The Transwheel's unique design contains eight free-turning rollers perpendicular to the axle arranged around the Transwheel periphery.

This distinct design combined with the rotation of the wheel body provides the ability to move loads in any direction without "freezing." Its enclosed, light-weight body is self cleaning and keeps out foreign material.

Transwheels are constructed with a standard ABS body, Nylon rollers, a Polyethylene center axle, Delrin sprockets, and stainless steel roller axles with low friction coatings. The combination of these materials creates a corrosion-resistant wheel that allows steam cleaning, water immersion, and outdoor use. Self-lubricating properties eliminate the need for oil or grease.

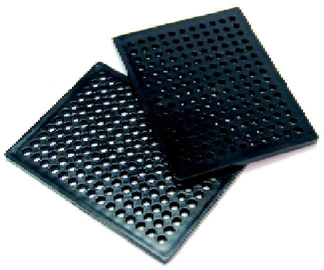
It includes free roller 2 rows. Each row includes 8 pieces of roller. With these rollers, this wheel can move both forward and sideways easily. When 3 wheels are attached to the base. ROBO128 moves in all directions with little friction. Robo-128 can move any direction with a variant of speeds by using these wheels.

Key technical features of this wheel are follows :

- 2-Inches (49.2 mm.) diameter
- Come with metal coupling for BO2 DC motor gearbox.
- Center bore diameter 3/8 Inches (9.5mm.)
- Roller is made by nylon. Available 8 pieces per row.
- Weight 1.75 oz. or 49.61 gram (1 oz = 28.35 gram) not include the metal coupling
- Load rating 11.3 kilogram or 25 pound

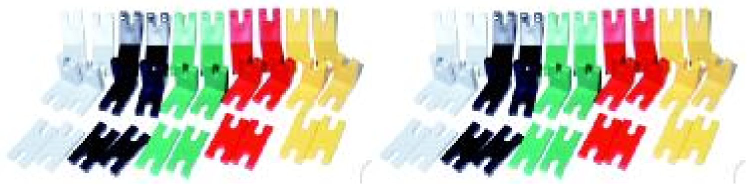
1.5.2 Plastic grid plate set

Includes each of the universal plastic grid palte 2 sizes; 80x60mm. and 80x80mm. Each plate provides 3mm. diameter holes with 5mm. pitch.



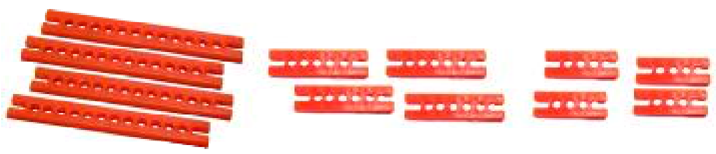
1.5.3 Plastic joiners

60 pieces of varied color joiners made from PVC plastic. They can be connected together or by using screws and 3 mm nuts in installation. There are 4 types; Right angle, Obtuse, Straight joiner and Hole straight joiner.



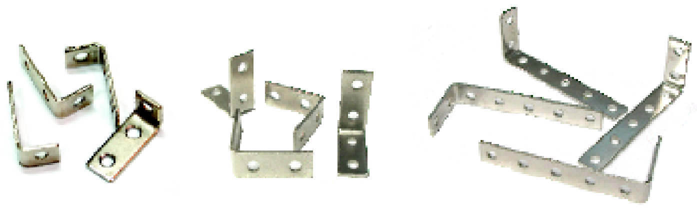
1.5.4 Strip joiners

They are made from plastic. Each joiner has 3mm. hole 5mm. pitch. Each joiner can be connected together for length expansion. There are 4 pieces of 3 sizes; 3, 5 and 12 holes type. Total 12 pieces.



1.5.5 Right-angle metal shaft

It is 7.5mm. width right-angle metal shaft. Each shaft has 3mm. hole for inserting the screw to fix with another structures. The set includes 4 pieces of 1x2, 2x2 and 2x5 holes metal shaft.



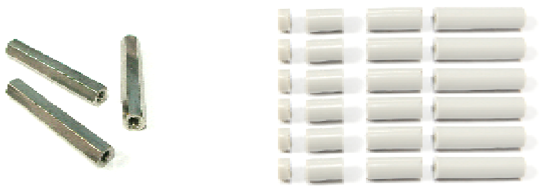
1.5.6 Screw and Nut set

Includes 4 of 3x8mm. M3 screws, 30 of 3x10mm. M3 screws, 4 of 3x15 mm. M3 screws, 4 of 3x40mm.M3 screws, 10 of 3x8mm. flat-head screws and 30 of 3mm. M3 nuts.

1.5.7 Metal and plastic spacer

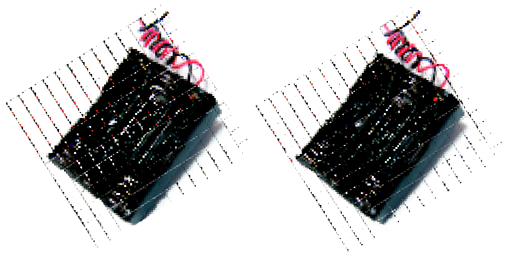
There are metal parts for supporting the plate and sensor board. They are made from nickel plated metal. Includes 6 of 33mm. metal hexagonal spacers. Each standoff has 3mm. thread through-hole.

There are some mechanical parts for supporting the plate and sensor board. This kit includes 4 pieces set of plastic spacer (3mm., 10mm., 15mm. and 25mm.) 4 sets



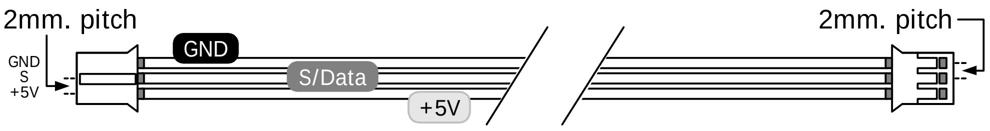
1.5.8 Battry holder

It is 2 of 3AA battery holder series connection. The positive and negative wire are available.



1.5 .9 JST3AA-8 cable

This is an INEX standard cable, 3-wires combined with 2mm. The JST connector is at each end. 8 inches (20cm.) in length. Used for connecting between controller board and all sensor modules in the Robo-128 kit. The wire assignment is shown in the diagram below.



Chapter 2
Robo-128 Development tools

Robo-128 supports all developent tools including assembly, basic and C language. However, we suggest using the C/C++ programming tool for Robo-128. It is called Wiring (*www.wiring.org.co*). Wiring is an open source programming environment and electronics i/o board for exploring the electronic arts, tangible media, teaching and learning computer programming and prototyping with electronics. It illustrates the concept of programming with electronics and the physical realm of hardware control which are necessary to explore physical interaction design and tangible media aspects.

Wiring is an open project initiated by Hernando Barragan (Universidad de Los Andes Architecture and Design School). Wiring started at the Interaction Design Institute Ivrea in Italy and it is currently developed at the Universidad de Los Andes in Colombia.

Wiring builds on Processing, an open project initiated by Ben Fry (Broad Institute) and Casey Reas (UCLA Design Media Arts). Processing evolved from ideas explored in the Aesthetics and Computation Group at the MIT Media Lab.

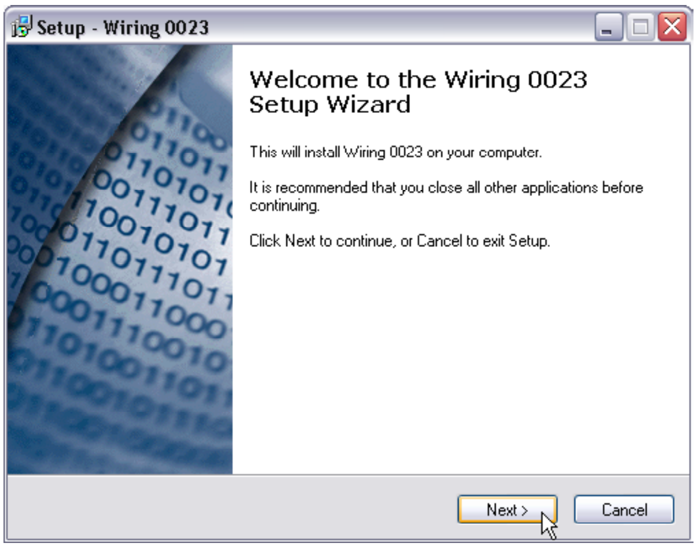
The operation system that supported is :

- Mac OS X 10.4 (Both Intel and PowerPC CPU)
- Windows XP and Vista
- Fedora Core 6 (Linux)

This chapter describes about introduction to Wiring. Begin with Installation, explain about Wiring IDE components and Menu bar details.

2.1 Software installation

(1) Insert CD-ROM of Robo-128 to CD-ROM drive. Click on the file *Wiring0023 Setup.exe.* The welcome window is appeared.



Welcome to the Wiring 0023 Setup Wizard

This will install Wiring 0023 on your computer.

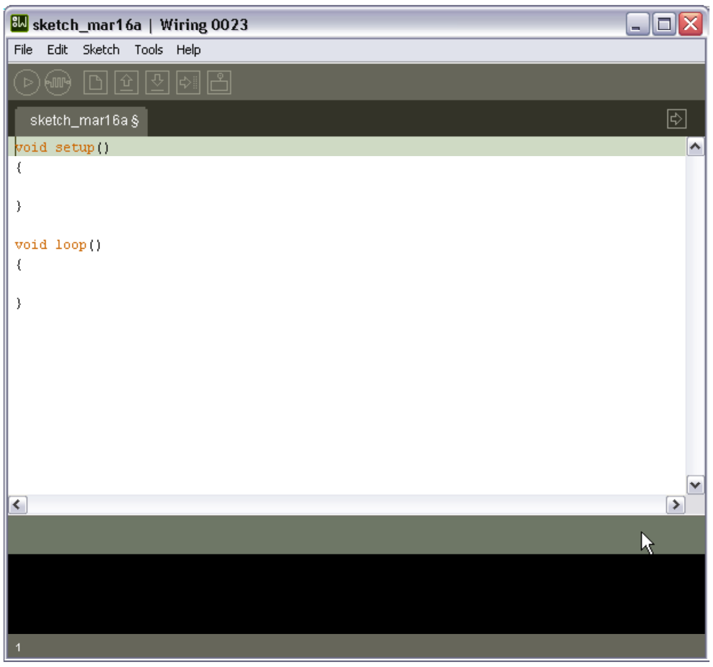
It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.

(2) Click on the OK button in each intallation step until installation is completed.

(3) This installation includes Wiring software and USB driver for Robo-128 controller board.

(4) Run software by select menu Start **à** All Programs **à** Wiring **à** Wiring .



```
void setup()
{

}

void loop()
{

}
```

2.2 Checking the USB Serial port for Robo-128

After installing Wiring IDE and USB driver, the next step is to check the USB serial port or virtual COM port that was created by USB driver.

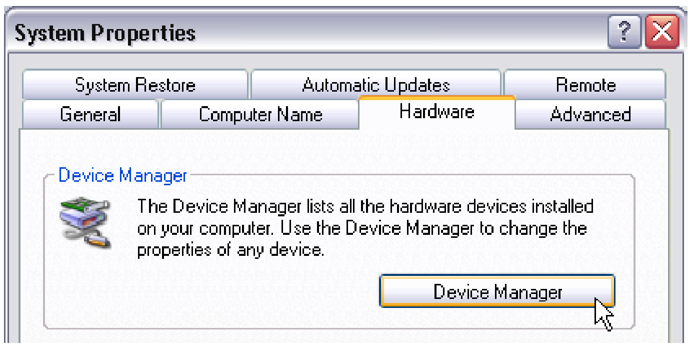
(1) Plug the USB cable between Robo-128 with USB port and turn-on

(2) Wait until USB indicator is on.

(3) Click on the Start button and select to Control Panel

(4) Double-click on System icon

(5) Select tab Hardware and click on the Device Manager button.



The Device Manager lists all the hardware devices installed on your computer. Use the Device Manager to change the properties of any device.

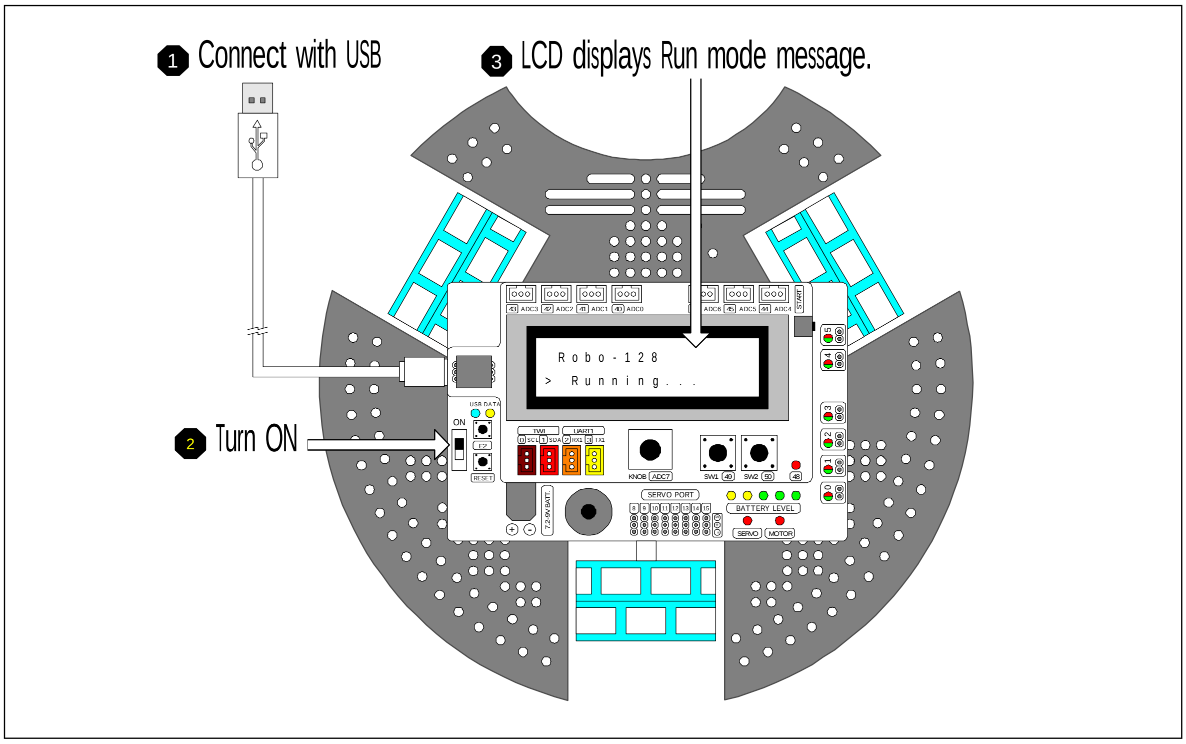
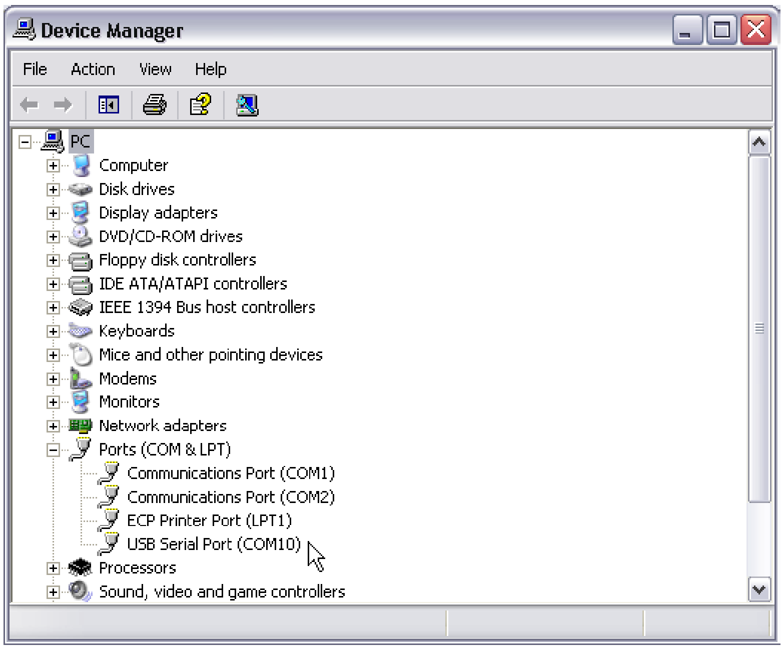


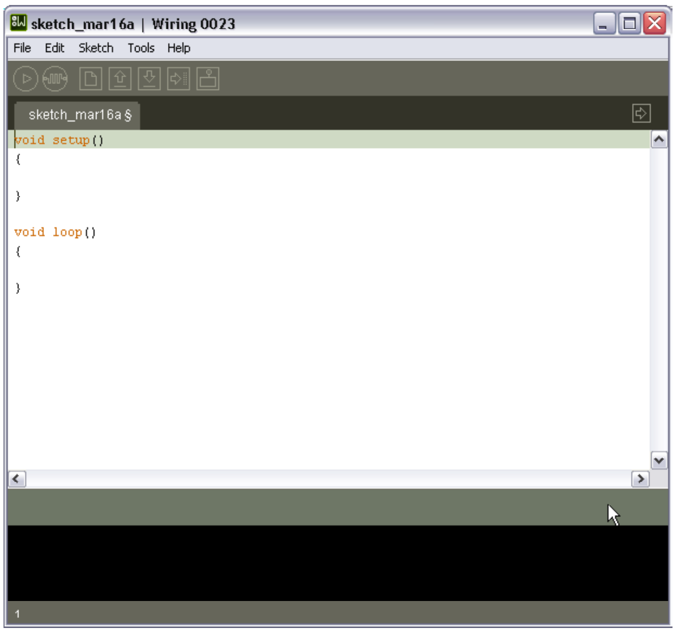
Figure 2-1 : Checking USB serial port of Robo-128

(6) Check the hardware listing at Port. You should see USB Serial port . Check the position. Normally it is COM3 or higher (for example; COM10). You must use this COM port with the Wiring IDE software.



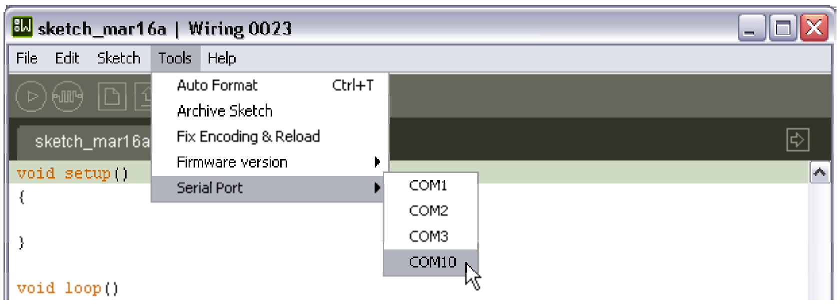
2.2 Robo-128 with Wiring IDE interfacing

(1) Open Wiring IDE by double-click on the file *wiring.exe* in the folder *Wiring-0023* (number of version could be changed) Wait for a while. The main window of Wiring IDE will appear.



```
void setup()
{

}

void loop()
{

}
```

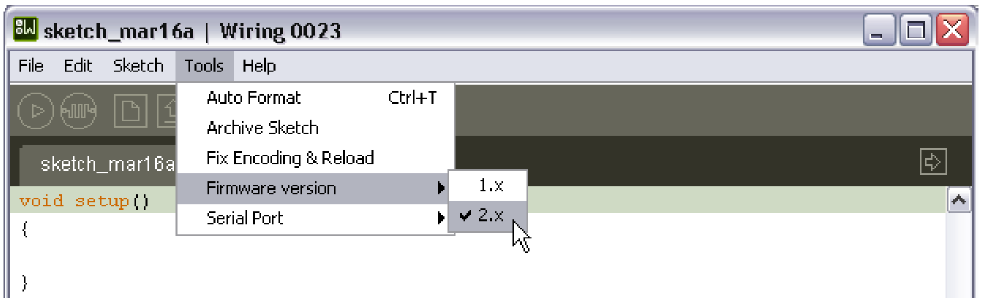
(2) Select menu Tools à Serial Port to choose the USB serial port of Robo-128. It is COM10



```
void setup()
{

}

void loop()
```

Must do this step for every new connection of the Robo-128 with Wiring IDE

(3) Select the bootloader firmware of Wiring I/O hardware which is connected with Wiring IDE by select menu Tools à Firmware version à 2.x



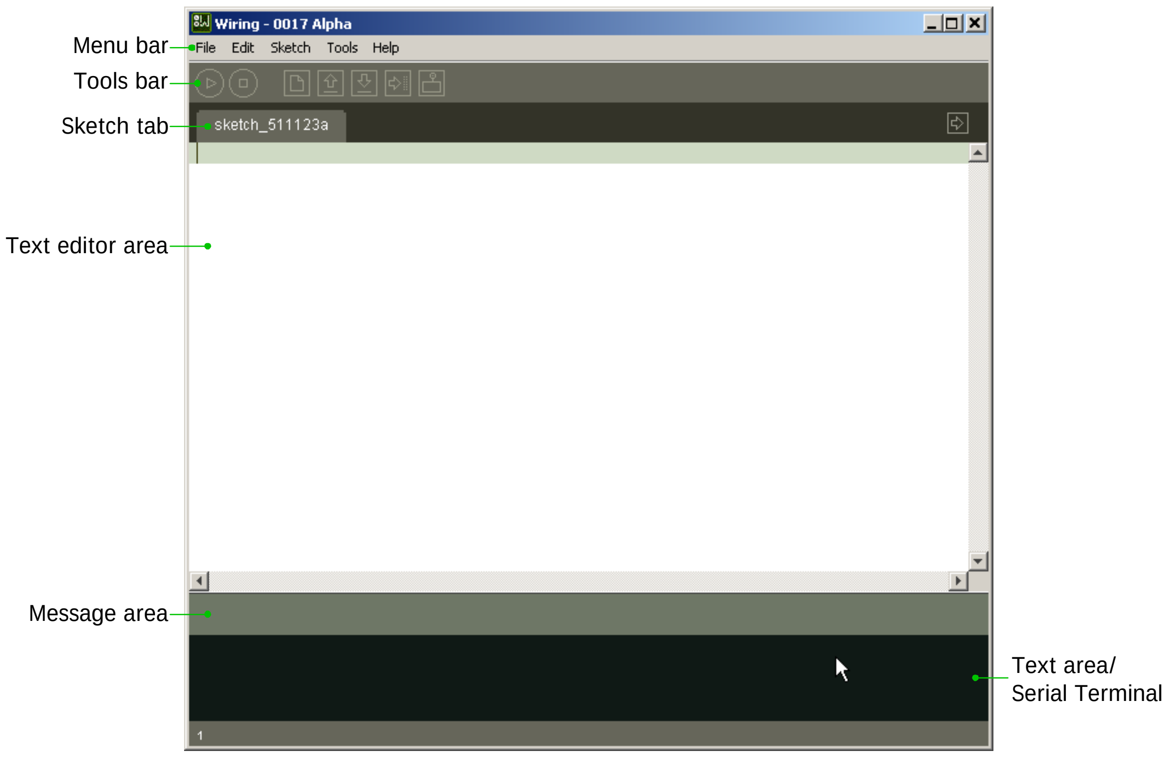
```
void setup()
{

}
```

Now Robo-128 is ready for interfacing and code development with the Wiring IDE.

Chapter 3
Wiring IDE and Sketch uploading

Wiring is Open Source Software. The PDE (Processing Development Environment) is released under the GNU GPL (General Public License). The export libraries (also known as 'core') are released under the GNU LGPL (Lesser General Public License).

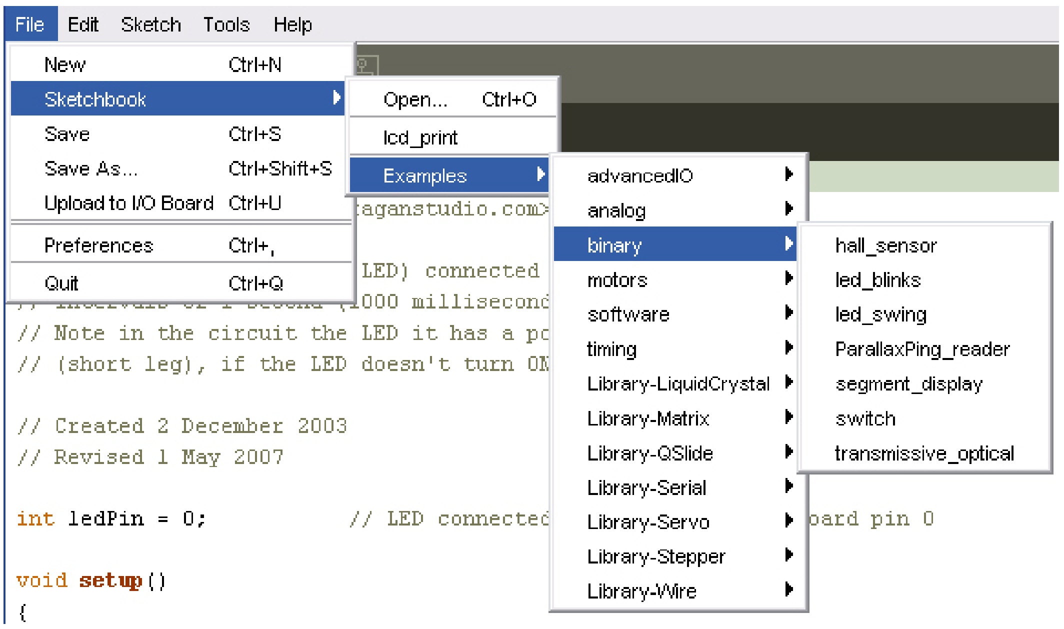
Wiring IDE is designed a simple and usable editor for writing and running programs. The Wiring Environment (Integrated Development Environment or IDE) has a text editor and compiler for writing programs for the Wiring I/O Board. See the figure below. When the "run" button is clicked, the program compiles. The toolbar provides functionality for compiling programs, creating a new sketch, opening, saving, and exporting to the I/O Board. Additional commands are available through the menus. The message area gives feedback while saving and exporting and also shows the locations of errors when programs are compiled. The text area presents messages and can be written from the I/O Board using with the print() programming function.



3.1 Menu bar

It includes the File, Edit, Sketch, Tools and Help menu. It works with the active current sketch tab only.

3.1.1 File



```
aganstudio.com>
LED) connected
1000 millisecond
// Note in the circuit the LED it has a po
// (short leg), if the LED doesn't turn ON

// Created 2 December 2003
// Revised 1 May 2007

int ledPin = 0;           // LED connected          oard pin 0

void setup()
{
```

New (Ctrl+N) or click on the button on the Tools bar : Creates a new sketch, named as the current date is the format “sketch_YYMMDDa” such as sketch_080407a.

Open (Ctrl+O) or click on the button on the Tools bar. : Gives the option to open a sketch from anywhere on the local computer or network, the sketchbook, or to open an example.

Save (Ctrl+O) or click on the button on the Tools bar : Saves the open sketch in it’s current state.

Save as... (Ctrl+Shift+O) : Saves the currently open sketch, with the option of giving it a different name. Does not replace the previous version of the sketch.

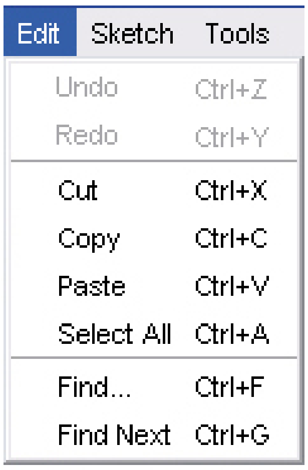
Upload to I/O Board (Ctrl+U) or click on the button on the Tools bar. : Be default, exports the program to the Wiring I/O Board (inthis document is Robo-128 controller board). After the files are exported, the directory containing the exported files is opened. There is more information about uploading below.

Preferences : Allows you to change some of the ways Wiring works.

Quit (Ctrl+Q) : Exits the Wiring Environment and closes all Wiring windows.

3.1.2 Edit

The Edit menu provides a series of commands for editing the Wiring files.



Undo (Ctrl+Z) : Reverses the last command or the last entry typed. Cancel the Undo command by choosing Edit ? Redo.

Redo (Ctrl+Y) : Reverses the action of the last Undo command. This option is only available, if there has already been an Undo action.

Cut (Ctrl+X) : Removes and copies selected text to the clipboard (an off-screen text buffer)

Copy (Ctrl+C) : Copies selected text to the clipboard.

Paste (Ctrl+V) : Inserts the contents of the clipboard at the location of the cursor, and replaces any selected text.

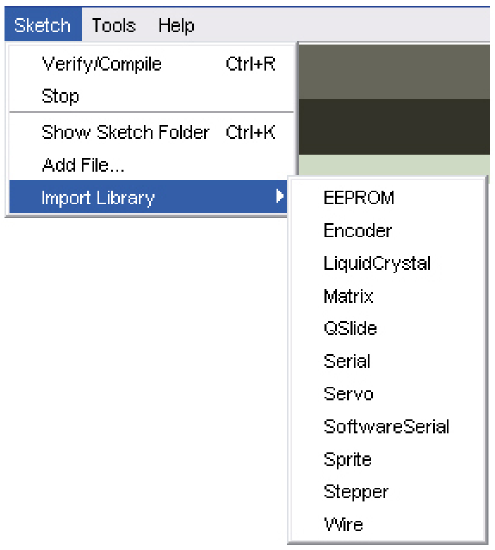
Select All (Ctrl+A) : Selects all of the text in the file which is currently open in the text editor.

Find (Ctrl+F) : Finds an occurrence of a text string within the file open in the text editor and gives the option to replace it with a different text.

Find Next (Ctrl+G) : Finds the next occurrence of a text string within the file open in the text editor.

3.1.3 Sketch

The Sketch menu provides a series of commands relate compiling the Wiring files.



Verify/Compile (Ctrl+R) or click on the ▷ button on the Tools bar : Verify the code (compiles the code)

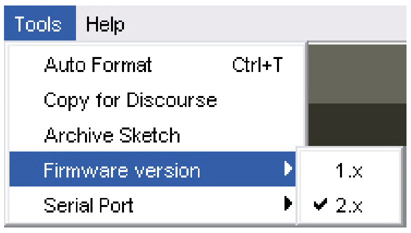
Stop or click on the □ button on the Tools bar : Stops current activity.

Show Sketch Folder : Opens the directory for the current sketch.

Add File : Opens a file navigator. Select a code files to add it to the sketches "data" directory.

Import Library : Open the included library of Wiring.

3.1.4 Tools

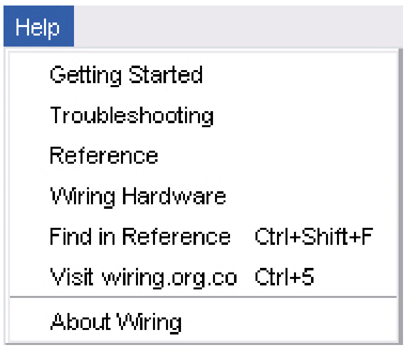


Auto Format (Ctrl+T) : Attempts to format the code into a more human-readable layout. Auto Format was previously called Beautify.

Firmware version : Select the suitable bootloader firmware of Wiring I/O board.

Serial Port : Allows to select which serial port to use as default for uploading code to the Wiring I/O Board or monitor data comming from it. The data coming from the Wiring I/O Board is printed in character format in the text area region of the console.

3.1.5 Help



Getting Started : Opens the getting started in the default Web browser.

Troubleshooting : Opens the troubleshooting in the default Web browser.

Reference : Opens the reference in the default Web browser. Includes reference for the language, programming environment, libraries, and a language comparison.

Find in Reference (Ctrl+Shift+F) : Select a word in your program and select “Find in Reference” to open that reference HTML page.

Visit Wiring.org.co (Ctrl+5) : Opens default Web browser to the Wiring homepage.

About Wiring : Opens a concise information panel about the software.

0015 Alpha
Wiring
An open project initiated by Hernando Barragán
© 2004-2008 Hernando Barragán
Processing is © 2004-2008 Ben Fry and Casey Reas
© 2001-2004 Massachusetts Institute of Technology

3.2 Toolbar

The toolbar provides access to the seven basic functions of Processing: Run, Stop, New, Open, Save, Export, Serial monitor.

Run : Compiles the code

Stop : Terminates any activity on the editor

New : Creates a new sketch. In Wiring, projects are called sketches.

Open : Select and load a pre-existing sketch. A menu opens and you may choose from your own sketchbook, examples, or you can open a sketch from anywhere on your computer or network.

Save : Saves the current sketch into the Processing sketches folder. If you want to give the sketch a name other than the current date, you can choose Save As from the File menu.

Upload to I/O Board : Exports the current sketch into the sketchbook and uploads it to the Wiring I/O Board (inthis document is Robo-128 controller board). The directory containing the files is opened.

Serial monitor : Opens a serial port connection to monitor the data comming from the Wiring I/O Board, this is very useful for debugging and verification.

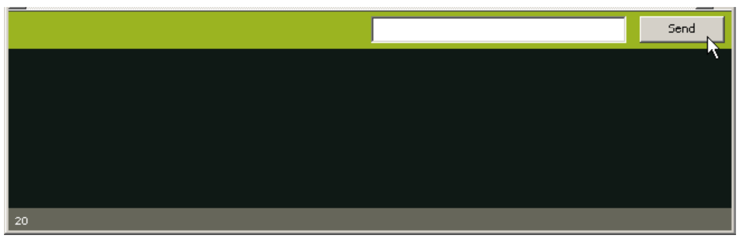
3.3 Serial monitor

Wiring IDE has a Serial monitor. It is a serial data communication tool. User can transmit, receive and show the serial data via this monitor with USB serial port of computer. In the developed sketch code, must put two imporatant commnands as follows :

1. Serial.begin() : Set the baud rate of serial data communication. Normally the baud rate value is 9600 bit per second. Must add this command into Setup() of sketchbook.

2. Serial.println() : Assign the sending message to Serial monitor on the Wiring IDE.

Openning the Serial monitor is very easy. Click on the button at Toolbar. The message area at the bottom of the main screen will change to the Serial monitor window following the figure below.



3.4 How to develop the program

(1) Check the hardware connection and set the Robo-128 to Program mode following figure 3-1. It is important that the USB serial port is now connected to Robo-128 controller board and your computer is running the Wiring IDE with all configurations ser properly.

(2) Open the Wiring IDE. Create the new sketch file by clicking on the button at Toolbar or select from File à New

(3) Type the example code below

```
#include <robot.h>                  // Include main library
int ledPin = 48;                    // LED connected to pin 48 (bootloader)
void setup()
{
    lcd("Hello Robot!");            // Title message on LCD
    pinMode(ledPin, OUTPUT);        // Sets the digital pin as output
}
void loop()
{
    digitalWrite(ledPin, HIGH);     // Sets the LED on
    delay(1000);                    // Waits for a second
    digitalWrite(ledPin, LOW);      // LED off
    delay(1000);
}
```

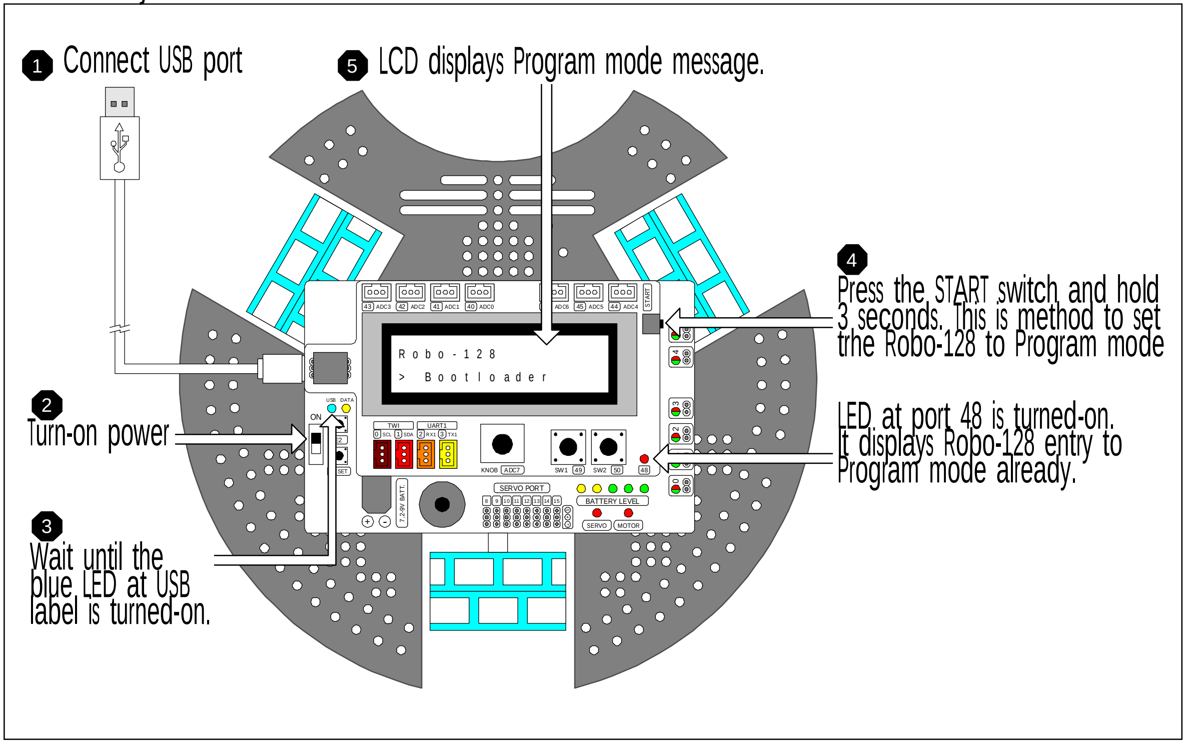
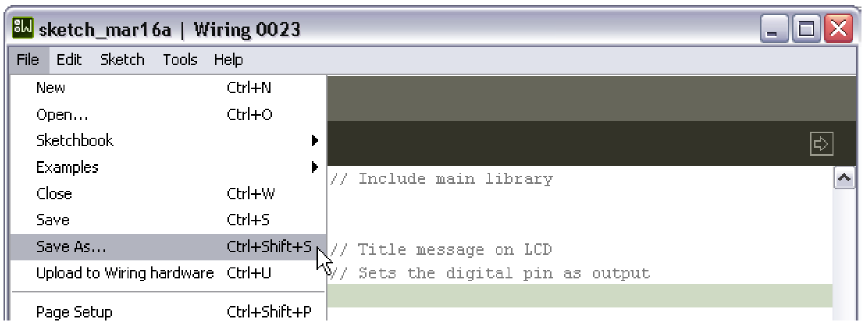

1 Connect USB port
5 LCD displays Program mode message.
4 Press the START switch and hold 3 seconds. This is method to set trhe Robo-128 to Program mode
LED at port 48 is turned-on. It displays Robo-128 entry to Program mode already.
2 Turn-on power
3 Wait until the blue LED at USB label is turned-on.
Robo-128
> Bootloader

Figure 3-1 : How to set the Robo-128 to Proigram mode

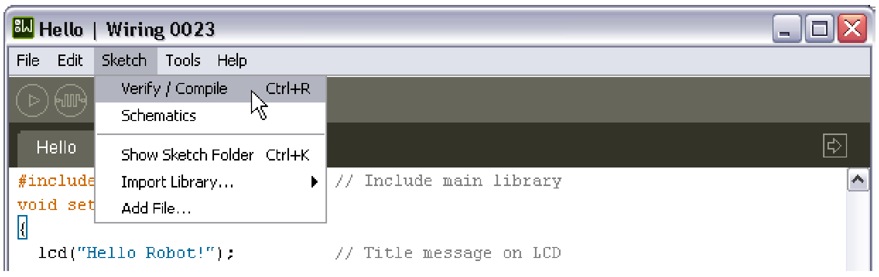
This example code demonstrates the Robo-128 basic operation. An LCD module displays testing message and LED at port 48 is controlled.

(4) Go to menu File à Save As for saving the new sketch as hello. Now we have *hello.pde* in the *hello* folder.



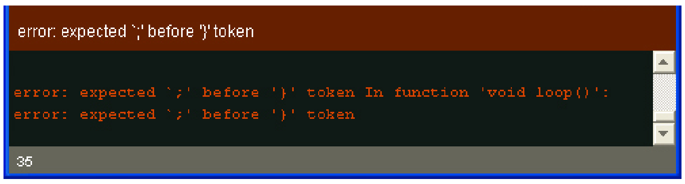
```
// Include main library


// Title message on LCD
// Sets the digital pin as output
```

(5) Compile this sketch file by clicking on the button at Toolbar ot menu Sketch à Compile/Verify.



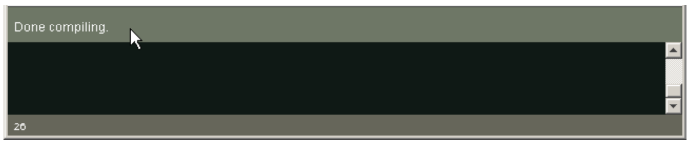
```
#include                          // Include main library
void set
{
  lcd("Hello Robot!");            // Title message on LCD
```

If have any error occurs, the error and warning message are shown at the Message area and Text area following the figure below.



```
error: expected `;' before ')' token

error: expected `;' before ')' token In function 'void loop()':
error: expected `;' before ')' token

35
```

Back to edit the code to fix any error and re-compile again until completely. The Message area shows Done compiling message following the picture below.



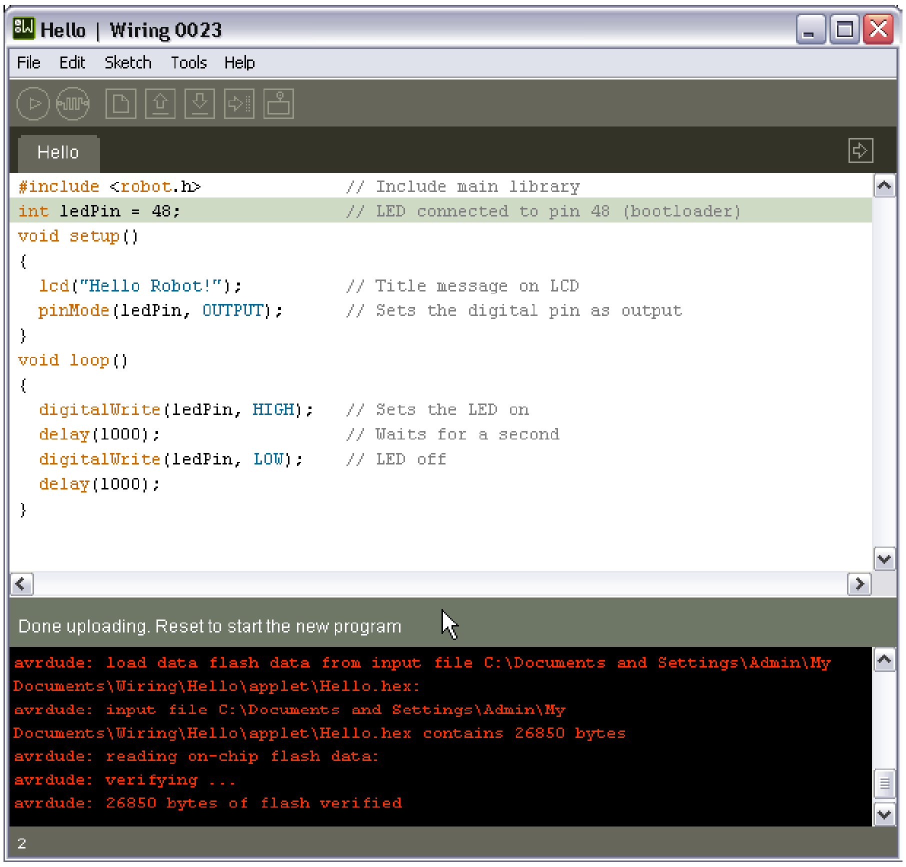
```
Done compiling.

26
```

After done compiling, in the *hello* folder have the new folder occur. It is *Applet* folder. This folder contains C/C++ source code and the output file .hex. for example as *hello.hex*, *hello.cpp* and *hello.pde*

(6) Connect the Robo-128 with USB port. Turn-on power. Wait until USB connection is completely (blue LED at USB is turned-on) .

(7) Press and hold the START button on theRobo-128 controller board 3 seconds. The LCD module shows Entry program mode message following the figure 3-1.

(8) Click on the Upload to Wiring Hardware. Code uploading is started. Wait until uploading complete. The message Done uploading. RESET to start the new program. is shown in the status bar of Wiring IDE.



```
#include <robot.h>              // Include main library
int ledPin = 48;                // LED connected to pin 48 (bootloader)
void setup()
{
  lcd("Hello Robot!");          // Title message on LCD
  pinMode(ledPin, OUTPUT);      // Sets the digital pin as output
}
void loop()
{
  digitalWrite(ledPin, HIGH);   // Sets the LED on
  delay(1000);                  // Waits for a second
  digitalWrite(ledPin, LOW);    // LED off
  delay(1000);
}
```

Done uploading. Reset to start the new program

```
avrdude: load data flash data from input file C:\Documents and Settings\Admin\My
Documents\Wiring\Hello\applet\Hello.hex:
avrdude: input file C:\Documents and Settings\Admin\My
Documents\Wiring\Hello\applet\Hello.hex contains 26850 bytes
avrdude: reading on-chip flash data:
avrdude: verifying ...
avrdude: 26850 bytes of flash verified
```

2

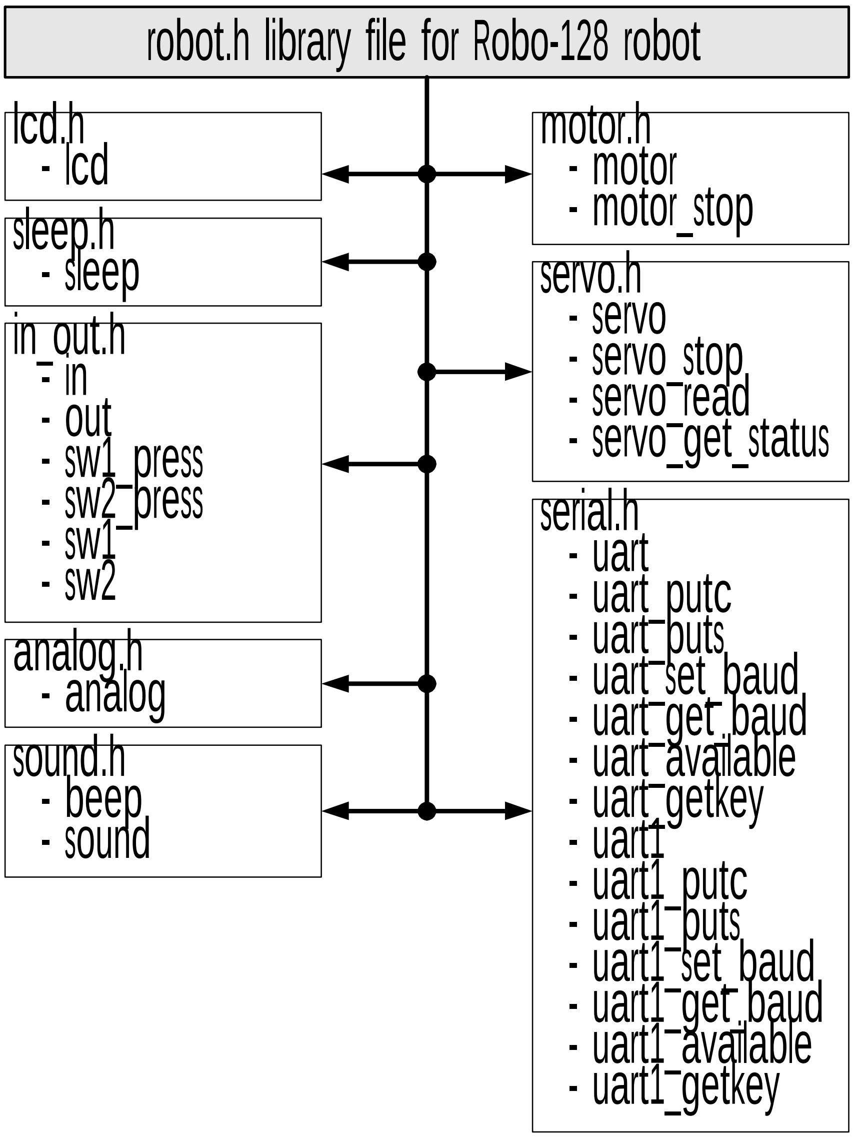
(9) Press the START switch at the Robo-128 controller board again. Robo-128 entry to Run mode. It executes the code.

LCD module displays message “Hello World!” and LED at port 48 is blinked.

Chapter 4
Robo-128 Library file

C/C++ program development with Wiring for Robo-128 is supported by robot.h library file. With this library, user can create the control program for Robo-128 easier and faster.

The sturcture of the robot.h library file is shown below.



4.1 lcd.h : LCD module library

This library file supports all instructions for displaying message and value at the LCD module on the Robo-128 controller board. This library must be included at the top of the program with the command #include as follows :

#include <lcd.h> or #include <robot.h>

Main function of this library is lcd

Syntax

```
void lcd(char *p,...)
```

Parameter

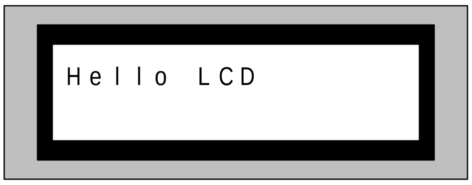
p - Type of display data. Support the special character for setting display method.

Command	*Operation*
%c or %C	Display 1 character
%d or %D	Display the decimal value -32,768 to +32,767
%l or %L	Display the decimal value -2,147,483,648 to +2,147,483,647
%f or %F	Display floating point 3 digits
#c	Clear message before next display
#n	Display message on the second line (bottom line)

Example 4-1

```
lcd("Hello LCD");  // Displays Hello LCD message at LCD module
```

Result :

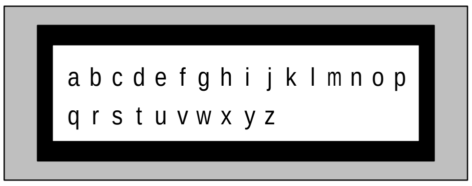


Example 4-2

```
lcd("abcdefghijklmnopqrstuvwxyz");
```

```
// Display string. If over 16 charactes, the next character will
// show on the second line automatically.
```

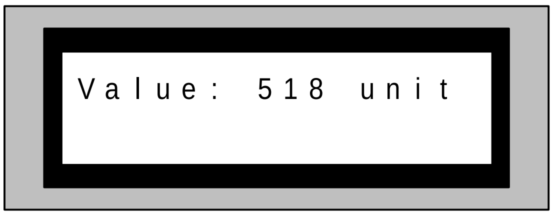
Result :



Example 4-3

```
lcd(“Value: %d unit “,518); // Display message with number date (518)
```

Result :

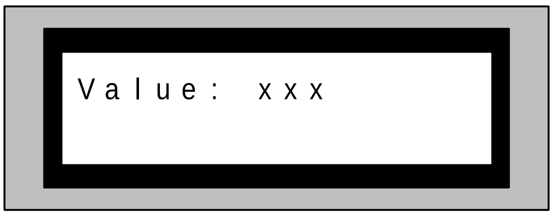


```
Value: 518 unit
```

Example 4-4

```
lcd(“Value: %d “,analog(4));
// Display analog value from anlog port 4 (PA4)
```

Result :



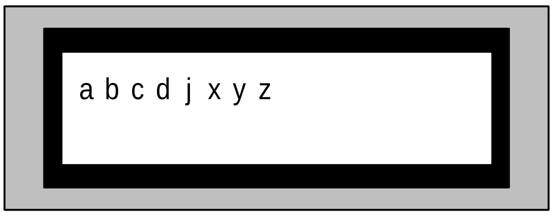
```
Value: xxx
```

therefore xxx as reading value 0 to 1023

Example 4-5

```
char c_test=’j’;
lcd(“abcd%cxyz”,c_test);
// Display character j with any message
```

Result :

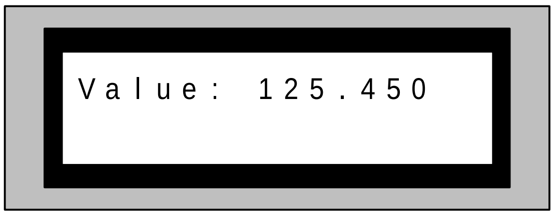


```
abcdjxyz
```

Example 4-6

```
lcd("Value: %f ",125.450);
// Display message with floating number 3 digit
```

Result :

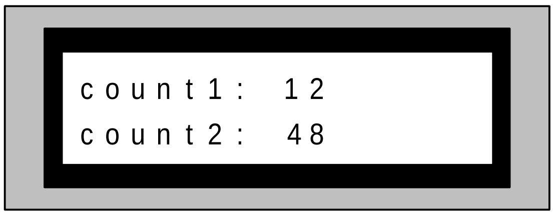


```
Value:  125.450
```

Example 4-7

```
lcd("count1: %d #ncount2: %d",12,48);
// Display message with 2 control code and special key #n
// for moving all message after #n to line 2 or bottom line of
// LCD screen
```

Result



```
count1:  12
count2:  48
```

4.2 sleep.h : Delay time library

This library file supports all instructions for time delaying. This library must be included at the top of the program with the command #include as follows :

#include <sleep.h> or #include <robot.h>

The important function is sleep . It delay time in millisecond unit.

Syntax

`void sleep(unsigned int ms)`

Parameter

ms - Set delay time in millisecond unit. The value is 0 to 65,535

Example 4-8

```
sleep(20);        // Dealy 20 miliisecond
sleep(1000);      // Delay 1 second
```

4.3 in_out.h : Input/Outp port library

This library file supports all instructions for readind and writing data to digital port of controller board. This library must be included at the top of the program with the command #include as follows :

```
#include <in_out.h> or #include <robot.h>
```

4.3.1 in

Read data from the specific digital port

Syntax

char in(x)

Parameter

x - Choose digital port number. it is 0 to 50

Return value

0 or 1

Example 4-9

```
char x;          // Declare x variable for keeping reading input data
x = in(49);      // Read port 49 and store data to x variable.
```

Example 4-10

```
char x;          // Declare x variable for keeping reading input data
x = in(50);      // Read port 50 and store data to x variable.
```

4.3.2 out

Write or send the data to the specific digital port

Syntax

out(char _bit,char _dat)

Parameter

_bit - Choose digital port number. it is 0 to 50

Example 4-11

```
out(43,1);       // Write port 43 with logic “1”
out(45,0);       // Write port 45 with logic “0”
```

4.3.3 sw1_press

This function loops to check the SW1 pressing. It returns value after switch is released.

Syntax

void sw1_press()

Example 4-12

```
................
sw1_press();         // Wait until the SW1 is pressed and released
................
```

4.3.4 sw2_press

This function loops to check the SW2 pressing. It returns value after switch is released.

<u>Syntax</u>

`void sw2_press()`

<u>Example 4-13</u>

```
..................
Sw2_press();  // Wait until the SW2 is pressed and released
.................
```

4.3.5 sw1

This function check the SW1 pressing in any time.

<u>Syntax</u>

`char sw1()`

<u>Return value</u>

"0" - SW1 is pressed

"1" - SW1 is not pressed

<u>Example 4-14</u>

```
char x;                 // Declare x  variable for keeping the value
x = sw1();              // Get SW1 status and store to x variable
```

4.3.6 sw2

This function check the SW2 pressing in any time.

<u>Syntax</u>

`char sw2()`

<u>Return value</u>

"0" - SW2 is pressed

"1" - SW2 is not pressed

<u>Example 4-15</u>

```
char x;                 // Declare x  variable for keeping the value
x = sw2();              // Get SW2 status and store to x variable
```

4.4 analog.h : Analog port library

This library file supports all instructions for reading the analog input port of Robo-128 controller board. This library must be included at the top of the program with the command #include as follows :

#include <analog.h> or #include <robot.h>

4.4.1 analog

This gets digital data from the analog to digital converter module of any analog port; ADC0 to ADC7.

Syntax

```
unsigned int analog(unsigned char channel)
```

Parameter

channel - Analog input (ADC0 to ADC7)

Return value

Digital data from analog to digital converter module. The value is 0 to 1023 (in decimal)

4.4.2 knob

This function gets data from ADC7 port. This port isconnected with variable resistor on-board. It is called KNOB.

Syntax

```
unsigned int knob()
```

Return value

Digital data from analog to digital converter module. The value is 0 to 1023 (in decimal)

Example 4-16

```
int val=0;          // Declare variable to keep the converted data
val  = analog(2);   // Get data from analog input ch. 2 (ADC2)
                    // and store data to val variable.
```

Example 4-17

```
int val=0;          // Declare variable to keep the converted data
val  = knob();      // Get data from analog input ch. 7
                    // (ADC7 or KNOB) and store data to val
                    // variable
```

4.5 motor.h

This library file supports all instructions for driving and controlling 6 DC motor outputs of Robo-128 controller board. This library must be included at the top of the program with the command #include as follows :

```
#include <motor.h> or #include <robot.h>
```

4.5.1 motor

It is DC motor driving function.

Syntax

```
void motor(char _channel,int _power)
```

Parameter

_channel - DC motor output of Robo-128; value is 0 to 5

_power - Power output value; it is -100 to 100

If set _power as positive value (1 to 100); motor is driven one direction.

If set _power as negative value (-1 to -100); motor is driven opposite direction.

If _power as 0; motor is stop. This value is not recommended. Use motor_stop function to stop motor better.

Example 4-18

```
motor(1,60);      // Drive motor ch.1with 60% of maximum power
motor(1,-60);     // Drive motor ch.1with 60% of maximum power and turn back direction.
```

Example 4-19

```
motor(2,100);     // Drive motor ch.2 with maximum power
```

4.5.2 motor_stop

This function is driving off a motor or stop.

Syntax

```
void motor_stop(char _channel)
```

Parameter

_channel - DC motor output of Robo-128; value is 0 to 5 and all (for driving off all channels)

Example 4-20

```
motor_stop(1);       // Stop motor ch.1
motor_stop(4);          // Stop motor ch.4
```

Example 4-21

```
motor_stop(ALL);     // All motor are stop.
```

4.6 servo.h : Servo motor library

This library file supports all functions for controlling 8 servo motor outputs of Robo-128. This library must be included at the top of the program with the command #include as follows :

#include <servo.h> or #inclue <robot.h>

4.6.1 servo

This is function of setting servo motor position.

Syntax

```
void servo(unsigned char _ch,unsigned int _pos)
```

Parameter

_ch - Servo motor output 8 to 15

_pos - Position value 0 to 180

4.6.2 servo_stop

This is stop servo motor function.

Syntax

```
void servo_stop(char _ch)
```

Parameter

_ch - Servo motor output 8 to 15 and all for stop all channel.

4.6.3 servo_read

This is function of servo motro position reading.

Syntax

```
int servo_read(char _ch)
```

Parameter

_ch - Servo motor output 8 to 15

Return value

The current position of servo motor shaft

4.6.4 servo_get_status

This is servo motor output status checking function.

Syntax

```
int  servo_get_status(char _ch)
```

Parameter

_ch - Servo motor output 8 to 15

Return value

- “1” - The selected servo motor output is operated.
- “0” - The selected servo motor output is not operated.

4.7 sound.h : Sound library

This library file supports all functions for sound generating of Robo-128. This library must be included at the top of the program with the command #include as follows :

#include <sound.h> or #inclue <robot.h>

4.7.1 beep

It is beep sound generating function. The beep frequency is 500Hz and 100 millisecond duration time.

Syntax

void beep()

4.7.2 sound

This is programmable sound generating function.

Syntax

void sound(int freq,int time)

Parameter

freq - Set frequency with value 0 to 32,767

time - Set duration time in millisecond unit from 0 to 32,767

Example 4-22

```
beep();              // Drives beep sound with 100 millisecond duration
sound(1200,500);     // Drives sound with 1200Hz 500 millisecond
```

4.8 serial.h : Serial data communication library

This library file supports all functions for sending and receiving the serial data via UART port of Robo-128. This library must be included at the top of the program with the command #include as follows :

#include <serial.h> or #include <robot.h>

4.8.1 Hardware connection

UART0 port

UART0 port is connected via USB to Serial converter chip; FT232RL. For connecting with computer, must connect via USB port on Robo-128 controller board. This connector is same port for downloading.

UART1 port

Connect via RXD1 (port 2) and TXD1 (port 3)

4.8.2 uart

This is serial data sending function via UART0 port. The default baudrate is 115,200 bit per second.

Syntax

void uart(char *p,...)

Parameter

p - Type of data. Support the special character for setting display method.

Command	*Operation*
%c or %C	Display 1 character
%d or %D	Display the decimal value -32,768 to +32,767
%l or %L	Display the decimal value -2,147,483,648 to +2,147,483,647
%f or %F	Display floating point 3 digits
\r	Set the message left justify of the line
\n	Display message on the new line

4.8.3 uart_set_baud

This is baud rate setting function for UART0.

Syntax

void uart_set_baud(unsigned int baud)

Parameter

baud - Baud rate of UART0 2400 to 115,200

Example 4-23

```
uart_set_baud(4800);     // Set baud rate as 4,800 bit per second
```

4.8.3 uart_available

This is receiveing data testing function of UART0.

Syntax

unsigned char uart_available(void)

Return value

- "0" : no data received
- more than 0 : received character

Example 4-24

```
char x =uart_available();
// Check the recieving data of UART0.
// If x value is more than 0; it means UART0 get any data.
// Read it by using uart_getkey function in the order next immediately.
```

4.8.4 uart_getkey

This is data reading function from receiver's buffer of UART0

Syntax

```
char uart_getkey(void)
```

Return value

- "0" : no data received
- data : received character in ASCII code

Example 4-25

```
#include <robot.h>                    // Get function
void setup()
{
}
void loop()                           // Main loop
{
    if(uart_available())              // Check incoming data
    {
        if(uart_getkey()=='a')        // Is key 'a' pressed ?
        {
            lcd("Key a Active!");     // Display message when get 'a' key already
            sleep(1000);              // Delay 1 second
        }
        else
        {
            lcd("#c");                // Clead display
        }
    }
}
```

Note : Default baud ratre of UART library is 115,200 bit per second. Data format is 8-bit and no parity.

4.8.5 uart1

This is serial data sending function via UART1 port. The default baud rate is 9,600 bit per second.

Syntax

void uart1(char *p,...)

Parameter

p - Type of data. Support the special character for setting display method. See details in uart0 function.

4.8.6 uart1_set_baud

This is baud rate setting function for UART1.

Syntax

void uart1_set_baud(unsigned int baud)

Parameter

baud - Baud rate of UART0 2400 to 115,200

Example 4-26

```
uart1_set_baud(19200); // Set baud rate as 19,200 bit per second
```

4.8.7 uart1_available

This is receiving data testing function of UART0.

Syntax

unsigned char uart1_available(void)

Return value

- “0” : no data received
- more than 0 : received character

Example 4-27

```
char x =uart1_available();      // Check the receiving data of UART1.
```

4.8.8 uart1_getkey

This is data reading function from receiver’s buffer of UART1.

Syntax

char uart1_getkey(void)

Return value

- “0” : no data received
- data : received character in ASCII code

4.9 Digital compass library

It is compass.h file. This library file is not included in robot.h library file. Must include the specific library file before using.

This library file supports all functions for interfacing the HMC6352 digital compass of Robo-128. This library must be included at the top of the program with the command #include as follows :

```
#include <compass.h>
```

4.9.1 compass_read

This reads the angle of the HMC6352 digital compass.

Syntax

```
int compass_read()
```

Return value

Angle value 0 to 359 defree

4.9.2 compass_set_heading

This is reference angle setting function. With this function, the current angle that read from digital compass is set to 0 degree reference.

Syntax

```
void compass_set_heading()
```

4.9.3 compass_read_heading

This is reference angle reading function. Use this function after set the new reference angle from compass_set_heading function.

Syntax

```
int compass_read_heading()
```

Return value

- 1 to 180 : positive angle (clock wise direction) of digital compass.
- -1 to -180 : negative angle (Counter-Clockwise direction) of digital compass.

Chapter 5
Robo-128 hardware experiment

This chapter presents some basic experiment about the hardware of Robo-128. All experiments demonstrate all hardware operations. Users or developers must create the code by using Wiring IDE following all steps as follows :

(1) Open Wiring and create the new sketch file

(2) Type C/C++ code in editor area of the sketch.

(3) Compile code by clicking on ▷ button or select menu Sketch à Compile/ Verify

(4) Connect Robo-128 with USB port of computer. Turn-on the power. Wait for USB connection succesfully. The blue LED at USB position is turned-on.

(5) Select mode to Program mode by pressing START switch and hold 3 seconds.

Robo-128's LCD module displays message

```
Robo-128
> Bootloader
```

and the red LED at port 48 is turned-on.

(6) Upload the sketch by clicking on ⇨ button or select at menu File à Upload to Wiring hardware

(7) Wait until the uploading success and no nay error. Press START switch again to set the operation to Run mode. Robo-128 operates following the code is uploaded.

Experiment 1 : Display message on LCD module

Experiment 1.1 - Simple message displaying

L1.1.1Create a new sketch file and save it as lcd_01. Type in the code following the Listing L1-1.

L1.1.2 Compile and upload the sketch file to Robo-128 robot.

L1.1.3 Run the code.

Robo-128's LCD shows message Hello Robot!

```
#include <robot.h>
void setup()
{
  lcd("Hello Robot!");    // Display message on LCD module
}
void loop()
{
}
```

Listing L1-1 : lcd_01.pde file; the C/C++ code of Wiring for testing LCD module display of Robo-128

Experiment 1.2 - 2 lines message displaying on LCD module

L1.2.1Create a new sketch file and save it as lcd_02. Type in the code following the Listing L1-2.

L1.2.2 Compile and upload the sketch file to Robo-128 robot.

L1.2.3 Run the code.

Robo-128's LCD shows message

Line1

Line2

```
#include <robot.h>
void setup()
{
 }
void loop()
{
  lcd("Line1#nLine2");    // Display message on LCD module
}
```

Listing L1-2 : lcd_02.pde file; the C/C++ code of Wiring for testing 2-line LCD module display of Robo-128

Experiment 1.3 - Message and number displaying on LCD module

L1.3.3 Create a new sketch file and save it as lcd_03. Type in the code following the Listing L1-3.

L1.3.2 Compile and upload the sketch file to Robo-128 robot.

L1.3.3 Run the code.

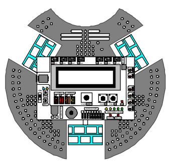
Robo-128's LCD shows message

```
Count:  xxx
```

The xxx *value is increasing every second.*

```
#include <robot.h>
int i = 0;              // Declare variable to keep the counter value
void setup()
{
}
void loop()
{
  lcd("Count: %d ",i);  // Display counter value at LCD module
  sleep(1000);          // Delay 1 second
  i++;                  // Increase counter
}
```

Listing L1-3 : lcd_03.pde file; the C/C++ code of Wiring for testing LCD module display of Robo-128 to display message and number value.



Experiment 2 : Using SW1 and SW2 of Robo-128

Experiment 2.1 - Using SW1 to start counter

This experiment demonstrates using SW1 of Robo-128 to start the counter and display value on LCD module.

L2.1.1 Create a new sketch file and save it as switch_01. Type in the code following the Listing L2-1.

L2.1.2 Compile and upload the sketch file to Robo-128 robot.

L2.1.3 Run the code.

Robo-128's LCD shows message

```
SW1 Press!
```

L2.1.4 Press the SW1 and release.

Program starts the counter and displays value on LCD module as follows :

```
Count: xxx
```

Therefore xxx *is counter value that increase every second.*

```
#include <robot.h>
int i=0;                // Declare variable to keep the counter value
void setup()
{
  lcd("SW1 Press!");    // Display title message
  sw1_press();          // Wait for SW1 pressing
  lcd("#c");            // Clear previous message before display the
                        // next messsage
}
void loop()
{
  lcd("Count: %d ",i);  // Display counter value
  sleep(1000);          // Delay 1 second
  i++;                  // Increase counter
}
```

Listing L2-1 : switch_01.pde file; the C/C++ code of Wiring for testing SW1 of Robo-128 operation

Experiment 2.2 - Check SW1 and SW2 pressing anytime

L2.2.1 Create a new sketch file and save it as switch_02. Type in the code following the Listing L2-2.

L2.2.2 Compile and upload the sketch file to Robo-128 robot.

L2.2.3 Run the code.

Robo-128's LCD shows message

```
Count:  xxx
```

Therefore xxx *is counter value that start with 10.*

L2.2.4 Press the SW1 and release.

Counter increase and displays value on LCD module everytime to press SW1.

L2.2.5 Press the SW2 and release.

Counter decrease and displays value on LCD module everytime to press SW2.

```
#include <robot.h>
int i=10;                      // Declare variable to keep the counter value
                               // and set default to 10
void setup()
{
}
void loop()
{
  lcd("Count: %d ",i);         // Display counter value
  if(sw1()==0)                 // Check SW1 pressing
  {
    i++;                       // Increase counter
    sleep(200);                // Delay for switch debouncing
  }
  if(sw2()==0)                 //  Check SW2 pressing
  {
    i--;                       // Decrease counter
    sleep(200);                // Delay for switch debouncing
  }
}
```

<u>Code description</u>

Code starts at setup funtion and loops to check the SW1 and SW2 pressing. The condition checking is

1. If SW1 switch is pressed - sw1() function returns 0 value.

Code increses counter and store to i variable. Delay short time to decounce the swtich pressign effect.

2. If SW2 switch is pressed - sw2() function returns 0 value.

Code decreases counter and store to i variable.

Listing L2-2 : switch_02.pde file; the C/C++ code of Wiring for testing SW1 and SW2 of Robo-128 operation

Experiment 3 : Reading analog sensor

Experiment 3.1 - Reading KNOB position value

This experiment demonstrates reading value from KNOB; the on-board variable resistor of Robo-128 to display on LCD module. This device is connected with the ADC7 input of controller board.

L3.1.1 Create a new sketch file and save it as knob_01. Type in the code following the Listing L3-1.

L3.1.2 Compile and upload the sketch file to Robo-128 robot.

L3.1.3 Run the code.

Robo-128's LCD shows message

KNOB : xxx

therefore xxx *is KNOB position value; 0 to 1,023 as :*

- Turn the knob to last left position. Value is 0.

- Turn the knob to last right position. Value is 1023.

- Turn the knob to center position. Value is 512.

```
#include <robot.h>
void setup()
{
}
void loop()
{
  lcd("KNOB: %d    ",knob());   // Display KNOB value
  sleep(100);                   // Delay 0.1 second
}
```

Listing L3-1 : knob_01.pde file; the C/C++ code of Wiring for testing KNOB device of Robo-128

Experiment 3.2 - Using KNOB for counter mode setting

This experiment demonstrates how to use KNOB for mode selection and display counter value on the LCD module. If turn KNOB to left direction, the counter mode is count-down. If turn KNOB to right direction, counter is count-up mode.

L3.2.1 Create the new sketch file and save it as knob_02. Type in the code following the Listing L3-2.

L3.2.2 Compile and upload the sketch file to Robo-128 robot.

L3.2.3 Run the code.

Robo-128's LCD shows message

```
Count:  xxx
Count  Up
```

in Up-counter mode is selected

```
Count:  xxx
Count  Down
```

in Down-counter mode is selected

therefore xxx *is counter value. The default is 100.*

The counter mode is selected by turning the KNOB of Robo-128.

L3.2.4 Turn the KNOB from center to left direction

Counter is set to Down-counter mode. The value is decreased every second.

L3.2.5 Turn the KNOB from center to right direction.

Counter is set to Up-counter mode. The value is increased every second.

```
#include <robot.h>
int i=100;                // Declare variable to keep the counter value
                          // and set default to 10
int k;                    // Declare variable to keep value from KNOB
void setup()
{
}
void loop()
{
  lcd(“Count: %d    “,i); // Display counter value
  k = knob();             // Get KNOB value to store in k variable
  if(k>512)               // Check KNOB position over the right or not ?
  {
    i++;                  // If KNOB position over the right;
                          // increase value
    lcd(“#nCount Up    “);  // Display counter mode on the 2nd line of
                            // LCD module
  }
  else
  {
    i--;             // If KNOB position over the left; decrease value
    lcd(“#nCount Down  “);  // Display counter mode on the 2nd line of
                            // LCD module
  }
  sleep(1000);              // Delay 1 second
}
```

Code description

The condition of this code is

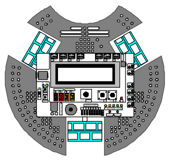
1. KNOB value is more than 512

Increase counter value in i variable and display “Count Up” for Up-counter mode.

2. KNOB value is less than 512 (execute else command)

Decrease counter value in i variable and display “Count Down” for Down-counter mode.

Listing L3-2 : knob_02.pde file; the C/C++ code of Wiring for using KNOB device of Robo-128 to select the counter mode



Experiment 4 : Action sound

Experiment 4.1 - Sound and switch

This experiment demonstrates the simple input/output port interfacing. The microcontroller gets the input from SW1 and SW2. When pressed, it generates a sound signal to drive to the piezo speaker on the controller board.

If SW1 is pressed, the 500Hz signal is generated at 0.1 second and 2000Hz (2kHz) signal is generated for 0.5 second if SW2 is pressed.

L4.1.1 Create a new sketch file and save it as sound_01. Type in the code following the Listing L4-1.

L4.1.2 Compile and upload the sketch file to Robo-128 robot.

L4.1.3 Run the code. Press the SW1 switch on the Robo-128 controller board.

The 500Hz signal is driven from the piezo speaker of the controller board at 0.1 second duration.

L4.1.4 Press the SW2 switch on the Robo-128 controller board.

The 2000Hz signal is driven from the piezo speaker of the controller board at 0.5 second duration.

```
#include <robot.h>
void setup()
{
}
void loop()
{
  if(sw1()==0)          // Check the SW1 pressing
  {
     beep();            // Generate 500Hz signal 0.1 second
     sleep(100);        // Delay for switch debouncing
  }
  if(sw2()==0)          // Check the SW2 pressing
  {
     sound(2000,500);   // Generate 2000Hz signal 0.5 second
     sleep(100);        // Delay for switch debouncing
  }
}
```

Listing L4-1 : sound_01.pde file; the C/C++ code of Wiring for driving the sound signal after the input switch input is pressed

Experiment 4.2 - Adjust frequency by using KNOB device

This experiment demonstrates an example about using the KNOB for adjusting the output signal frequency and displayed the frequency value on the LCD screen of Robo-128.

L4.2.1 Create a new sketch file and save it as sound_02. Type in the code following the Listing L4-2.

L4.2.2 Compile and upload the sketch file to Robo-128 robot.

L4.2.3 Run the code.

Robo-128's LCD shows message

```
Freq: xxx Hz
```

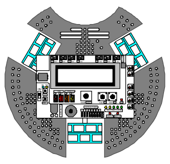
therefore xxx *is the frequency value that realte with KNOB adjustment*

L4.2.4 Adjust the KNOB on the controller board from left to right slowly.

The frequency value increases from 0 to 2046 Hz. Listen to the sound signal that increase according to the KNOB adjustment.

```
#include <robot.h>
int k;                  // Declare the variable to store KNOB value
int f;                  // Declare the variable to store the frequency
void setup()
{
}
void loop()
{
  k = knob();             // Read the KNOB value to store in k variable
  f = 2*k;                // Multiply by 2 for KNOB value
  lcd("Freq: %d Hz    ",f);
                          // Display the frequency value on LCD screen
  sound(f,200);           // Generate the sound siganl from f variable
                          // (It is 2 times of KNOB value) long 0.2
                          // second
  sleep(1000);            // Delay 1 second
}
```

Listing L4-2 : sound_02.pde file; the C/C++ code of Wiring for adjusting the sound frequency by using KNOB on the Robo-128 controller board



Ch. 0 and 1 turns in opposite directions

Experiment 5 : Motor control

Experiment 5.1 - Direction control

This experiment demonstrates the code for controlling the DC motor channel 0 and 1 to turn forward and backward every 3 seconds continuously.

Hardware connection

l Connect the first DC motor to DC motor output ch. 0 of Robo-128

l Connect the second DC motor to DC motor output ch. 1 of Robo-128

Procedure

L5.1.1 Create a new sketch file and save it as motor_01. Type in the code following the Listing L5-1

L5.1.2 Compile and upload the sketch file to Robo-128 robot.

L5.1.3Lift the robot over the floor and run the code.

Motor at Ch. 0 and 1 start and turn in opposite directions every 3 seconds.

```
#include <robot.h>
void setup()
{
}
void loop()
{
  motor(0,70);       // Drives the DC motor at ch. 0 with 70% of maximum power
  motor(1,70);       // Drives the motor at ch. 1 with 70% of maximum power
  sleep(3000);       // Delay 3 seconds before reverse direction

  motor(0,-70);      // Reverse direction and drives the DC motor at
                     // ch. 0 with 70% of maximum power
  motor(1,-70);      // Reverse direction and drives the DC motor at
                     // ch. 1 with 70% of maximum power
  sleep(3000);       // Delay 3 seconds before reverse direction again
}
```

Listing L5-1 : motor_01.pde file; the C/C++ code of Wiring for testing the motor’s direction control of the Robo-128 robot

Experiment 5.2 - Motor ON/OFF

This experiment demonstrates the simple motor control operation. Motor at ch. 0 and 1 will be controlled ON and OFF 3 swapping every 3 seconds continually. Still use same hardware connection from the experiment 5.1.

L5.2.1 Create a new sketch file and save it as motor_02. Type in the code following the Listing L5-2.

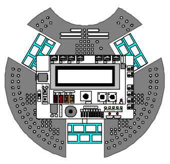
L5.2.2 Compile and upload the sketch file to Robo-128 robot.

L5.2.3Lift the robot over the floor and run the code.

Motor at Ch. 0 and 1 move and stop every 3 seconds.

```
#include <robot.h>
void setup()
{
}
void loop()
{
  motor(0,90);    // Drives the DC motor at ch. 0 with 90% of maximum power
  motor(1,90);    // Drives the DC motor at ch. 1 with 90% of maximum power
  sleep(3000);    // Delay 3 seconds
  motor_stop(0);  // Stop motor ch. 0
  motor_stop(1);  // Stop motor ch. 1
  sleep(3000);    // Delay 3 seconds
 }
```

Listing L5-2 : motor_02.pde file; the C/C++ code of Wiring for testing the motor's control of the Robo-128 robot



Experiment 6 : Servo motor control

Experiment 6.1 - Controls the servo motor's shaft position

Hardware connection

- Connect a standard RC servo motor with Servo motor output ch. 12 of Robo-128.

Procedure

L6.1.1 Create a new sketch file and save it as servo_01. Type in the code following the Listing L6-1.

L6.1.2 Compile and upload the sketch file to Robo-128 robot.

L6.1.3 Run the code.

Servo's motor shaft will turn between 60 and 120 degree position every 5 seconds

```
#include <robot.h>
void setup()
{
}
void loop()
{
  servo(12,60);          // Drives the servo motor at ch. 12 to
                         // turn the shaft to 60 degree position
  sleep(5000);           // Delay 5 seconds
  servo(12,120);         // Drives the servo motor at ch. 12 to
                         // turn the shaft to 120 degree position
  sleep(5000);           // Delay 5 seconds
}
```

Listing L6-1 : servo_01.pde file; the C/C++ code of Wiring for testing the servo motor control of the Robo-128 robot

Experiment 6.2 - Switch controlled servo motor position

This experiment demonstrates how to use SW1 and SW2 of Robo-128 controller to control servo motor's shaft position. SW1 is used for increasing the position and SW2 for deceasing the position. This experiment still requires the servo motor connection similar to the experiment 6.1

L6.2.1 Create a new sketch file and save it as servo_02. Type in the code following the Listing L6-1.

L6.2.2 Compile and upload the sketch file to Robo-128 robot.

```
#include <robot.h>
int p=90;           // Declare the servo motor position variable and
                    // set default as 90
void setup()
{
}
void loop()
{
  servo(12,p);      // Drive the servo motor at ch. 12 to destination
                    // that is defined by p variable
  lcd(“Servo: %d   “,p);
                    // Show the current position of servo motor’s shaft.
  if(sw1()==0)      // Check the SW1 pressingè
  {
    p++;            // Increase the servo postion
    if(p>180)       // The position is over 180 ?è
    {
      p=0;          // If over 180, restart to 0
    }
    sleep(100);     // Delay
  }
  if(sw2()==0)      // Check the SW2 pressingè
  {
    p—;             // Decrease the servo motor position
    if(p<0)         // The position is lower 0 ?è
    {
      p=0;          // Restart to 0 if position value lower than 0
    }
    sleep(100);     // Delay
  }
}
```

Listing L6-2 : servo_02.pde file; the C/C++ code of Wiring for servo motor position control of the Robo-128 robot

L6.2.3 Run the code.

Robo-128's LCD shows message

```
Servo:  xxx
```

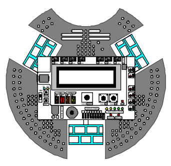
therefore xxx *is the current servo motor's shaft position in degree.*

L6.2.4 Press SW1 on the Robo-128 controller board.

Servo's motor shaft position is increased and the shaft turns to a new position. After the position increases to 180, it will then restarts at 0 again.

L6.2.5 Press SW2 on the Robo-128 controller board.

Servo's motor shaft position is decreased and turn the shaft turns to a new position. After the position decreases to a value lower than 0, it restarts at 0 again.



Experiment 7 : Using the ZX-IrEYE to detect the infrared ball

This experiment demonstrates how to detect the infrared ball with using the special sensor; ZX-IrEYE. It performs detection of direction and distance between the infrared ball and Robo-128 and displays the result on the LCD screen of the robot.

Note : The environmant infrared ray includes Sunlight, Incandescent lamp, flash light of any camera effect the operation of ZX-IrEYE sensor. User must consider about this effect before using this sensor.

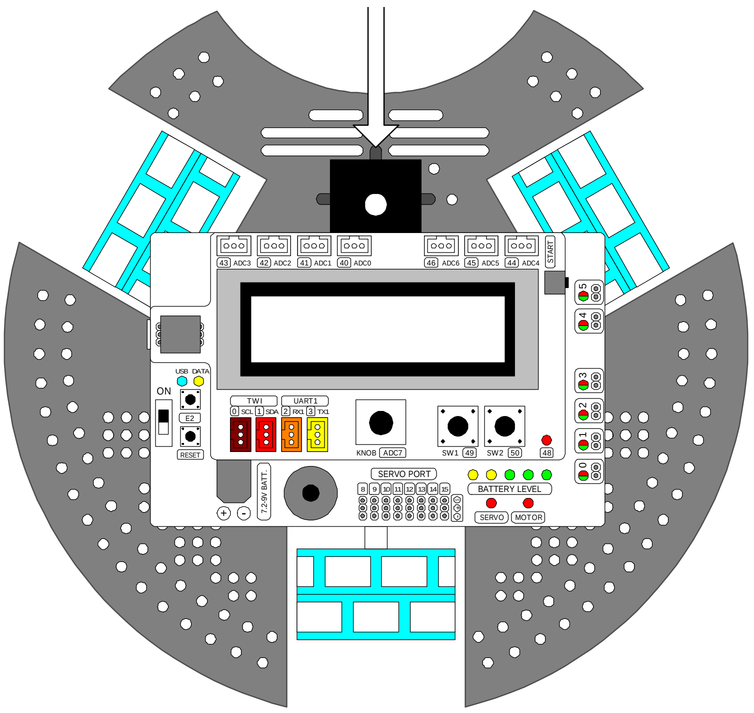
Hardware connection

l Connect the DIR output of ZX-IrEYE to ADC0 port of Robo-128

l Connect the RANGE output of ZX-IrEYE to ADC1 port of Robo-128

l Attach the ZX-IrEYE in front of the Robo-128. Turn the component side down to the robot chasis.

Attach the ZX-IrEYE at front of the robot and turn the component side down to the chasis.



```
#include <robot.h>
int dir;                  // Declare the Direction variable
int range;                // Declare the Range variable
void setup()
{
}
void loop()
{
  dir = analog(0);        // Get the direction value from ZX-IrEYE
  range = analog(1);      // Get the range value from ZX-IrEYE.
  lcd(“DIR: %d    #nRANGE: %d    “,dir,range);
     // Display the direction value on the upper line of LCD screen
     // Display the range value on the lower line of LCD screen
  sleep(100);             // Delay
}
```

Listing L7-1 : ball_sensor_01.pde file; the C/C++ code of Wiring for tesing the ZX-IrEYE; multi-direction infrared detector sensor

Procedure

L7.1 Create a new sketch file and save as ball_sensor_01. Type in the code following the Listing L7-1.

L7.2 Compile and upload the sketch file to Robo-128 robot.

L7.3 Run the code.

Robo-128's LCD shows message

DIR: xxx

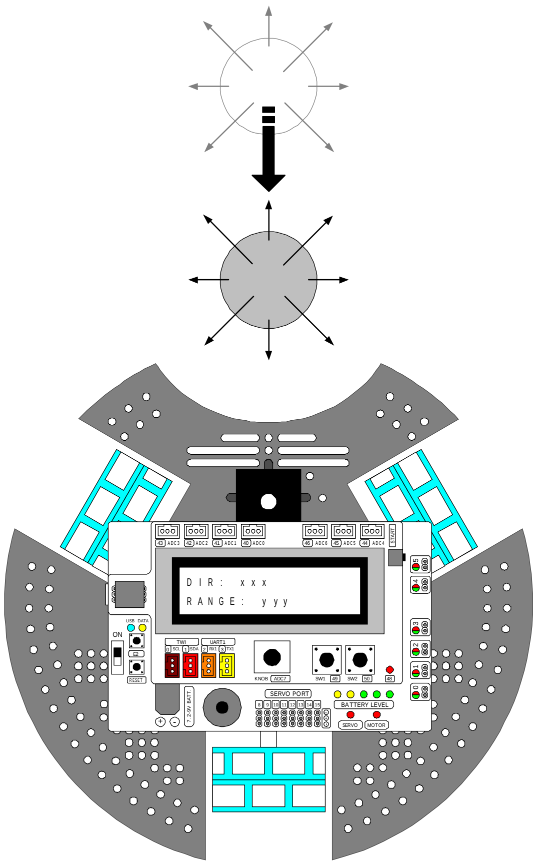
RANGE: yyy

therefore

xxx *is the direction tendency value of the infrared source. In this case is the infrared ball. The value is between 0 to 1023. In case attach the sensor with make face down, the sensor gives high value if the infrared ball locate at left side of sensor. It gives low value if the infrared ball locate at right side of sensor.*

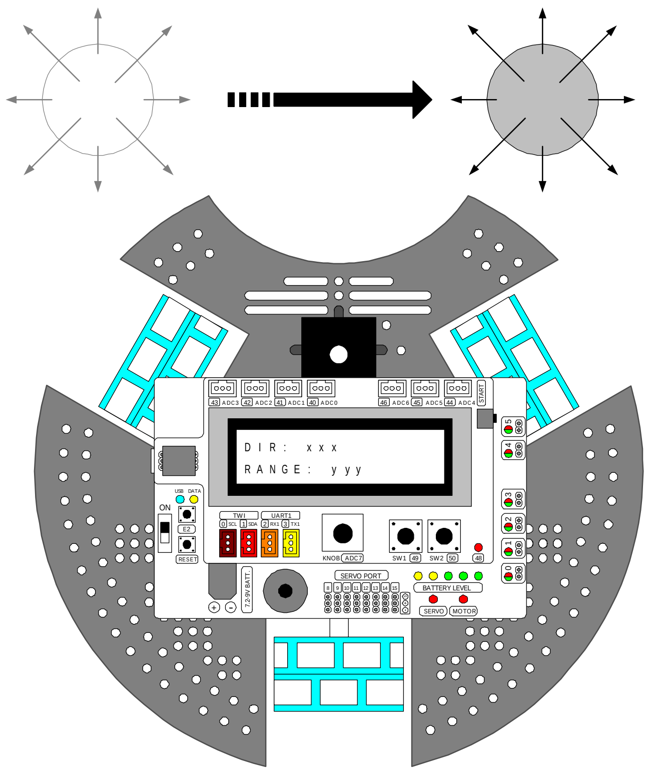
yyy *is the distance value between the infrared source and sensor. The value is 0 to 1023. The sensor gives high value when near the infrared source and decrease if the infrared source is far away.*

L7.4 Place the infrared ball in front of the Robo-128 about 30cm. Shift the ball to the robot slowly. Observe the robot's operation.

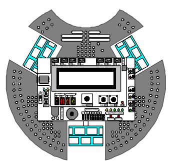


Range value is increasing when the ball closed-up sensor.

L7.5 Place the infrared ball at the front-left side of the Robo-128 about 30cm. away. Shift the ball to the front-right side slowly. Observe the robot's operation.



DIR (direction) value is decreasing when the ball is moved from left to right side.



Experiment 8 : Using the compass sensor

Experiment 8.1 - Read robot position

This experiment demonstrates about the reading position value from the HMC6532 digital compass of Robo-128 and display on LCD screen.

WARNING : The digital compass sensor includes the earth magnetic field sensor. It's performance is affected and the sensor can be damage if its placed near any strong external magnetic field such as Permanent magnet, Motor, etc...The user needs to consider this factor during the implementation of this sensor.

L8.1.1 Create a new sketch file and save as compass_01. Type in the code following the Listing L8-1.

L8.1.2 Compile and upload the sketch file to Robo-128 robot.

L8.1.3 Run the code.

Robo-128's LCD shows message

```
Angle: xxx
```

therefore xxx *is angle value 0 to 359*

L8.1.4 Turn around the robot slowly and observe the reading value on LCD screen

The angle is changed between 0 to 359

```
#include <robot.h>
#include <compass.h>         // Include the digital compass library
int angle;                   // Declare the compass angle variable
void setup()
{
}
void loop()
{
  angle = compass_read();    // Get the angle value from
                             //the digital compass sensor (0 to 359)
  lcd("Angle: %d    ",angle);// Display on the LCD screen
  sleep(100);                // Delay for displaying
}
```

Listing L8-1 : compass_01.pde file; the C/C++ code of Wiring for tesing the HMC6532 digital compass sensor module of the Robo-128

Experiment 8.2 - Set the reference angle

This experiment demonstrates the reading angle from setting the reference angle. This technique is important when require to set the custom reference angle. Normally the 0 degree of the digital compass is North pole. But in the soccer competition, robot must set the 0 degree to the front direction. This experiment can help and support this application.

L8.2.1 Create a new sketch file and save as compass_02. Type in the code following the Listing L8-2.

L8.2.2 Compile and upload the sketch file to Robo-128 robot.

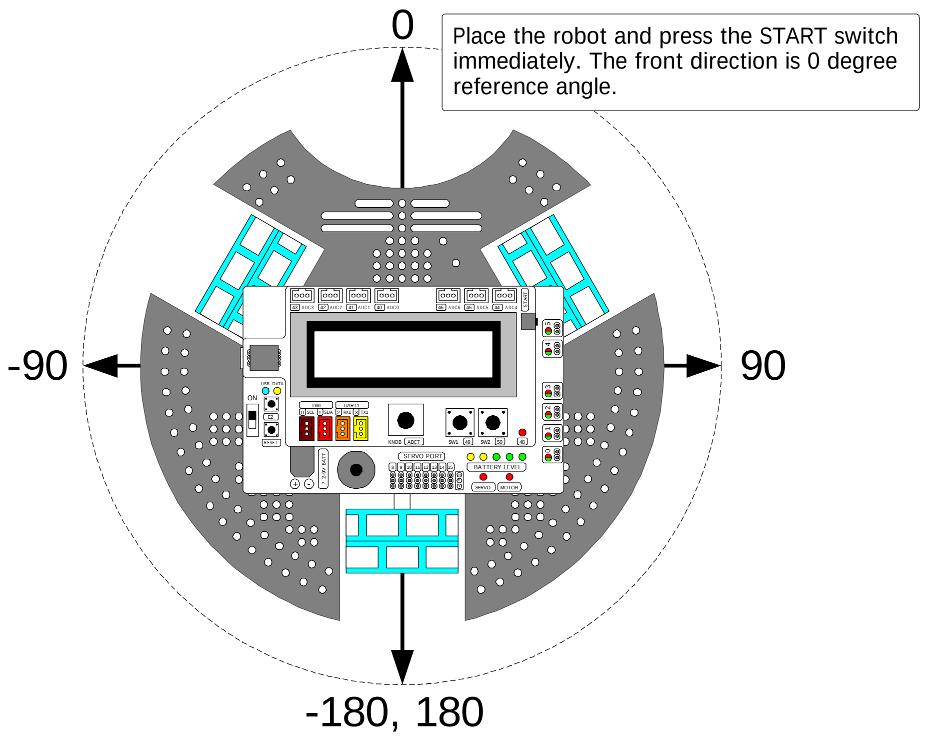
L8.2.3 Run the code.

Robo-128's LCD shows message

```
Angle: xxx
```

therefore xxx *is Angle value between -180 to 180 degree when turn around and refer the reference angle*

When the code starts, the current front direction of the Robo-128 is set to 0 degree.


0
Place the robot and press the START switch immediately. The front direction is 0 degree reference angle.
-90
90
-180, 180

```
#include <robot.h>
#include <compass.h>           // Include the digital compass library
int angle;                     // Declare the compass angle variable
void setup()
{
  compass_set_heading();       // Store the 0 degree reference angle
                               // for calculation the new angle
}
void loop()
{
  angle = compass_read_heading();
     // Get the angle value in format -180 to 180 degree
     // from the reference angle
  lcd("Angle: %d    ",angle);  // Display the reading angle
  sleep(100);                  // Delay for displaying
}
```

Listing L8-2 : compass_02.pde file; the C/C++ code of Wiring for setting the new reference angle of HMC6532 digital compass sensor

There is 2 cases of the reading angle from this technique to explain the robot position as follows :

Case 1

The sensor gives a value between 1 to 180 degree when deviated from the reference angle in a clock wise direction.

Case 2

The sensor gives a value between -1 to -180 degree when deviated from the reference angle in an anti-clock wise direction.

L8.2.4 Rotate the Robo-128 anti-clock wise direction (rotate left) slowly. Observe the result on LCD screen of Robo-128

The reading angle is changed between -1 to -180 degree.

L8.2.5 Rotate the Robo-128 clock wise direction (rotate right) slowly. Observe the result on LCD screen of Robo-128.

The reading angle is changed between 1 to 180 degree.

L8.2.6 Developers can set the new reference angle by placing the robot to the target direction and pressing the START switch again for reset the operation.

Chapter 6
Robo-128 movement

6.1 Robo-128 basic structure

Robo-128 has 3 omni-directional wheels and is attached at 60 degree angles following the figure 6-1. Each motor is driven by a DC motor driver circuit separately. User must create the program to control all motor driver circuits correctly for driving the robot.

The easiest way to move the Robo-128 is driving only one wheel. For example, drive the front-right wheel forward. Robo-128 will rotate with anti-clock wise direction. On the other hand, drive the front-right wheel backward. Robot will rotate with clock wise direction instead.

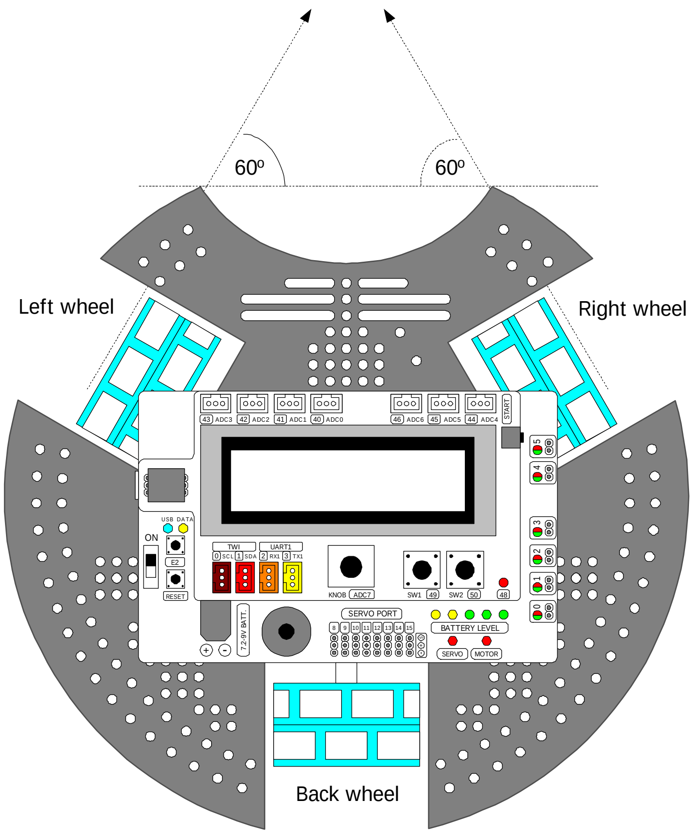
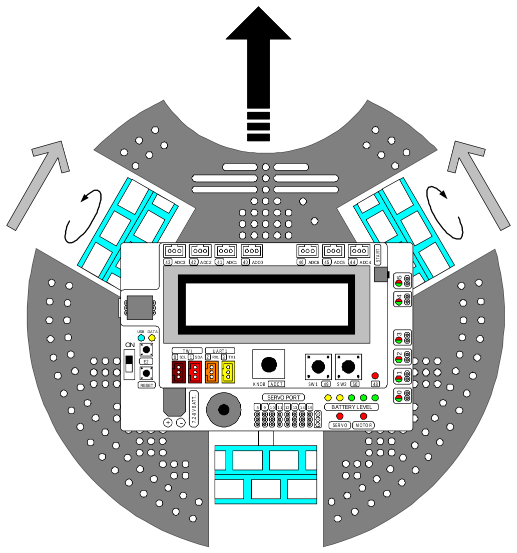


Figure 6-1 Wheel configuration of Robo-128

6.2 Robo-128 movement concept

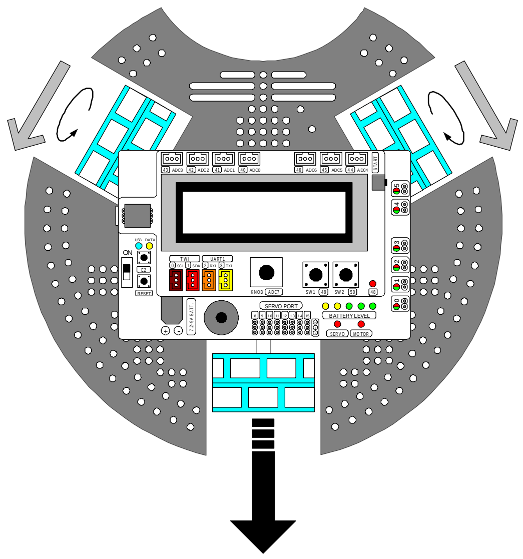
6.2.1 Move forward

Robo-128 moves forward by driving both front-left and front-right motors in forward direction with the same power.



6.2.2 Move backward

Robo-128 moves backward by driving both front-left and front-right motors in backward direction with the same power.



1. Drives the motor with front-left wheel backward

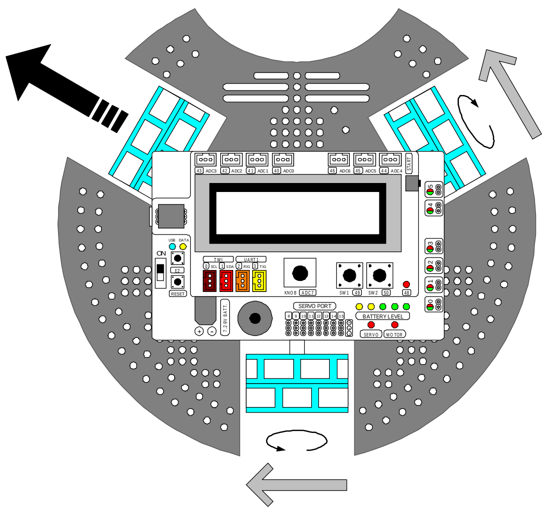
2. Drives the motor with back wheel to right direction

6.2.3 Move diagonal left

This movement has 2 direction; move diagonal left forward and backward.

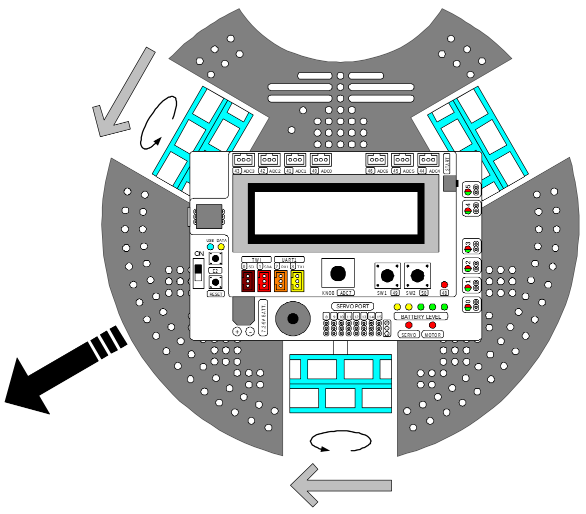
6.2.3.1 Move diagonal left forward

This movement use 2 drving wheels as front-right and back wheel. The front-right wheel is driven towards the forward direction. The back wheel is driven to the left direction. See the figure below. The black arrow represents the moving direction. The small grey arrow represents the wheel driving direction.



6.2.3.2 Move diagonal left backward

This movement usse 2 driving wheels as front-left and back wheel. The front-left wheel is driven towards the backward direction. The back wheel is driven to left direction. See the figure below. The driving power of both motors must be equal. If not, the robot will rotate instead The black arrow represents the moving direction. The small grey arrow represents the wheel driving direction.



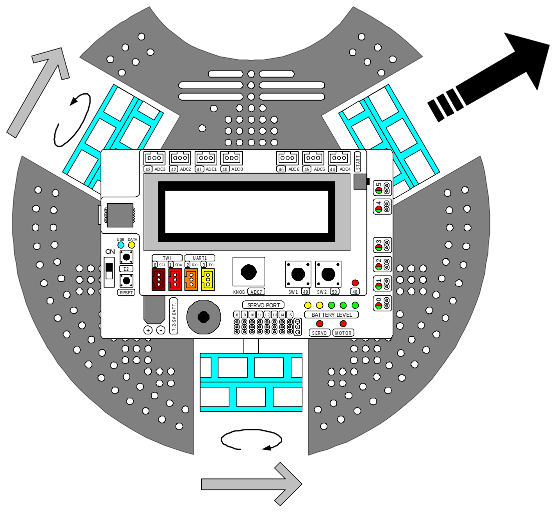
6.2.4.2 Move diagonal right backward

6.2.4 Move diagonal right

This movement has 2 direction; move diagonal right forward and backward.

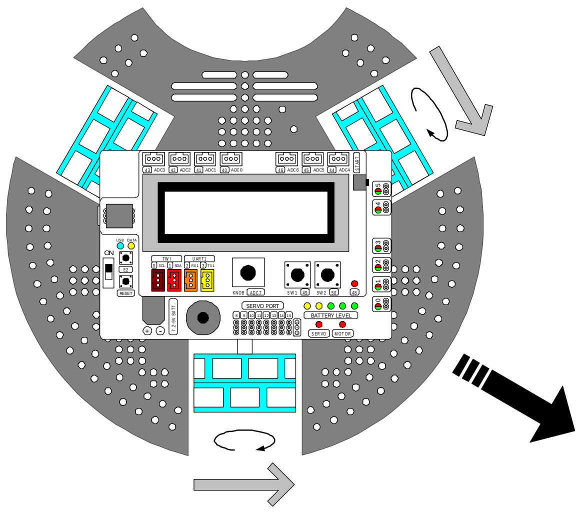
6.2.4.1 Move diagonal right forward

This movement use 2 drving wheels as front-left and back wheel. The front-left wheel is driven towards the forward direction. The back wheel is driven to right direction. See the figure below. The driving power of both motors must be equal. If not, the robot will rotate instead.



6.2.4.2 Move diagonal right backward

This movement use 2 drving wheels as front-right and back wheel. The front-right wheel is driven to backward direction. The back wheel is driven to right direction. See the figure below. The driving power of both motors must be equal. If not, the robot will rotate instead. The black arrow represents the moving direction. The small grey arrow represent wheel driving direction.

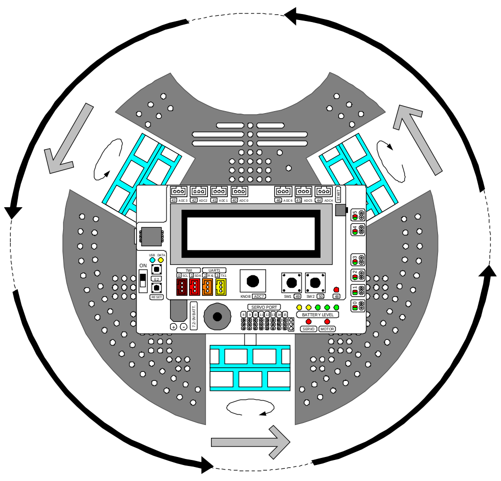


6.2.5 Spin left (anti-clockwise)

This movement uses 1 or 3 of wheels. The better performance is using 3 wheels. Each motor will be controlled with different methods as follows :

1. Drives the motor with front-left wheel backward
2. Drives the motor with back wheel to right direction
3. Drives the motor with front-right wheel forward

The figure below shows the operation of all motors for driving Robo-128 to rotate left.

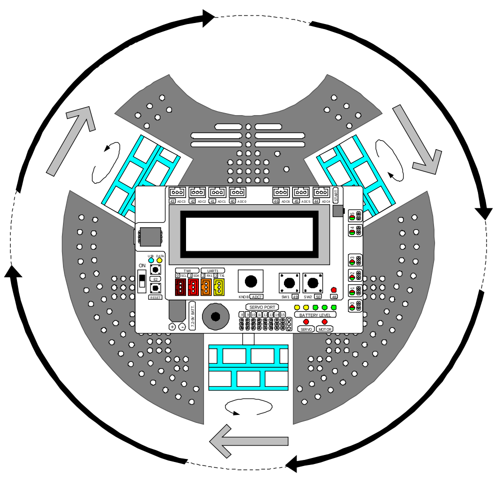


6.2.6 Spin right (clockwise)

This movement uses 1 or 3 of wheels same rotate left. The better performance is using 3 wheels. Each motor will be controlled with different methods as follows :

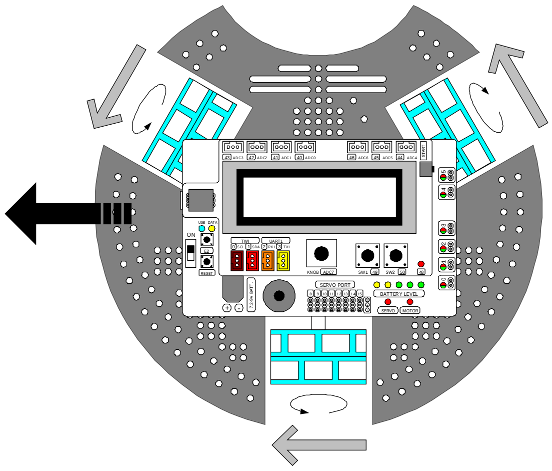
1. Drives the motor with front-left wheel forward
2. Drives the motor with back wheel to left direction
3. Drives the motor with front-right wheel backward

The figure below shows the operation of all motors for driving Robo-128 to rotate left.



6.2.7 Shift left

The figure below shows how to drive the Robo-128 to move left by sliding or shift left. All motors and wheels are driven together.



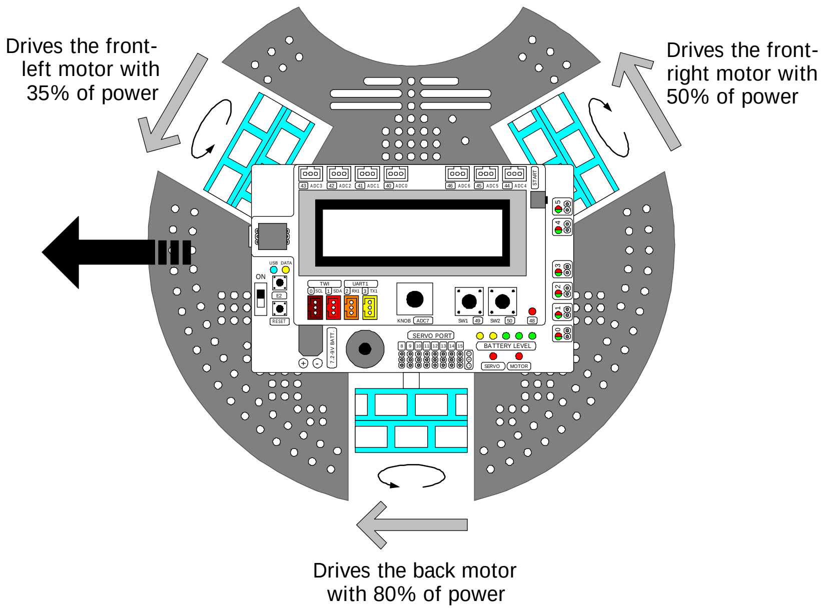
Each motor will be controlled with different methods as follows :

1. Drives the motor with front-left wheel backward by lowest power that compare with another wheels.

2. Drives the motor with back wheel to left direction using the highest power that compare with another wheels.

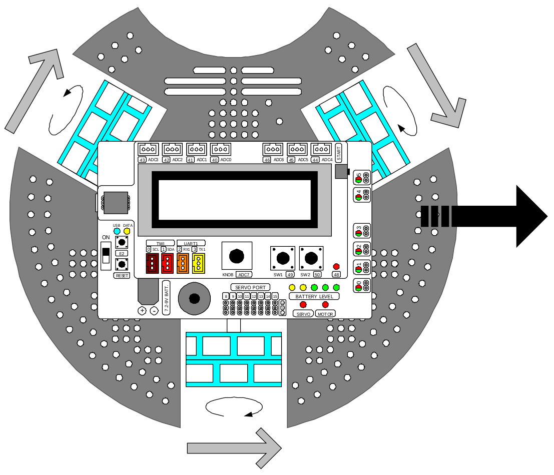
3. Drives the motor with front-right wheel forward using a larger power compared to the front-left wheel and lower than the back wheel.

The figure below shows the example of applying power to each motor of Robo-128 for shifting left.


Drives the front-left motor with 35% of power
Drives the front-right motor with 50% of power
Drives the back motor with 80% of power

6.2.8 Shift right

The figure below shows how to drive the Robo-128 to move right by sliding or shift right. All motors and wheels are driven together.



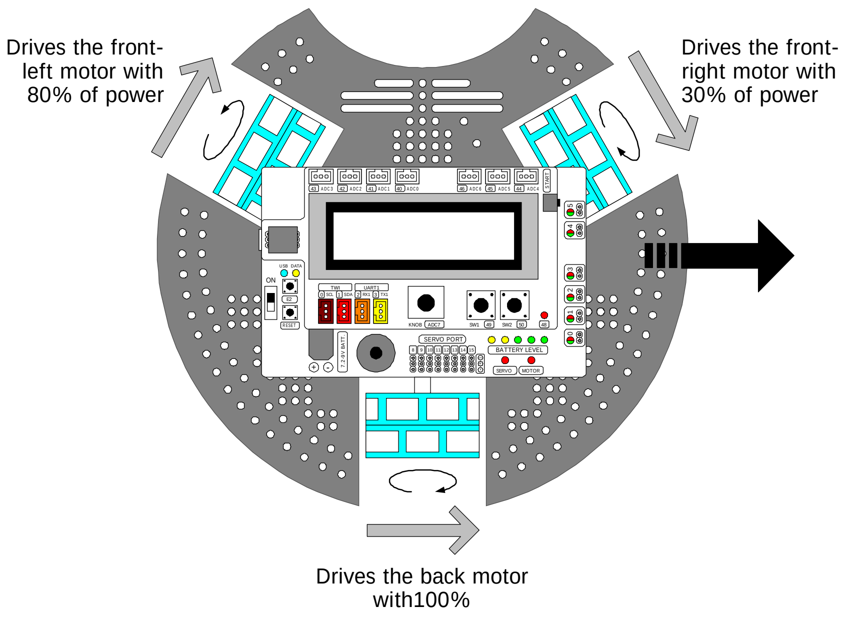
Each motor will be controlled with different methods as follows :

1. Drives the motor with front-left wheel forward using a larger power than the front-right wheel and lower than the back wheel.

2. Drives the motor with back wheel to right direction using the highest power that compare with another wheels.

3. Drives the motor with front-right wheel backward using the lowest power that compare with another wheels.

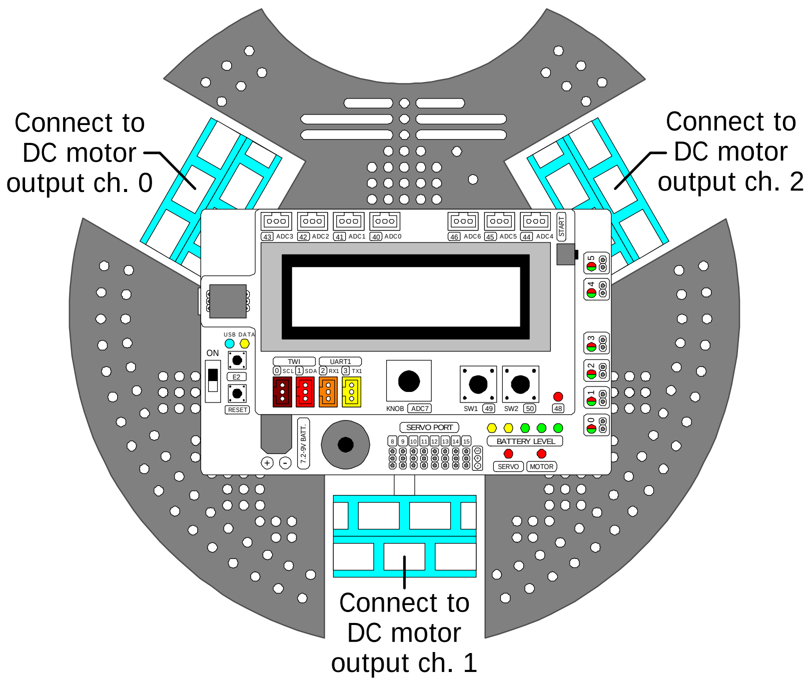
The figure below shows the example of applying power to each motor of Robo-128 for shifting right.


Drives the front-left motor with 80% of power
Drives the front-right motor with 30% of power
Drives the back motor with100%

Experiment 9 : Robo-128 simple movement

This is the motor configuration for Robo-128.

- Motor at the fron-left wheel is connected to DC motor output ch.0.
- Motor at the back wheel is connected to DC motor output ch.1.
- Motor at the fron-right wheel is connected to DC motor output ch.2.


Connect to DC motor output ch. 0
Connect to DC motor output ch. 2
Connect to DC motor output ch. 1

Experiment 9.1 - Wheel driving calibration with software

To test the wheel movements, we must set each motor turning. Any motor is driven with positive value from the motor function. Negative value turns the motor in an anti-clockwise direction. For example, motor at ch.1 is driven by positive value. It turns in a right direction. If it turns with left direction, it means the connection is incorrect. Remove the motor cable and invert the connection of the motor cable connector at ch. 1 output. You will need to test this with all the motors of the Robo-128.

L9.1 Unplug all motor cables from the motor output of Robo-128.

L9.2 Create a new sketch file and save it as motor_test. Type in the code following the Listing L9-1.

```
#include <robot.h>
void setup()
{
}
void loop()
{
  motor(0,80);
  motor(1,80);
  motor(2,80);
}
```

Code description

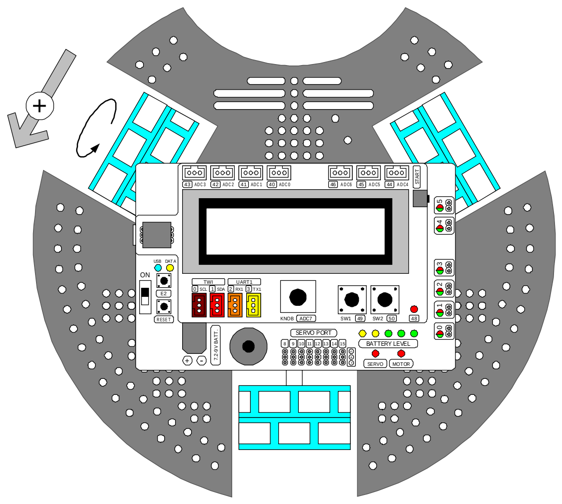
This code drive all motors by 80% of power. The motor voltage pole is shown by color of LED. If all motor get the positive power value, LED at each motor output is green. In the orther hands, LEd is red if get the negative value.

Listing L9-1 : motor_test.pde file; the C/C++ code of Wiring for testing the motor connection of Robo-128

L9.3 Compile and upload the sketch file to Robo-128 robot.

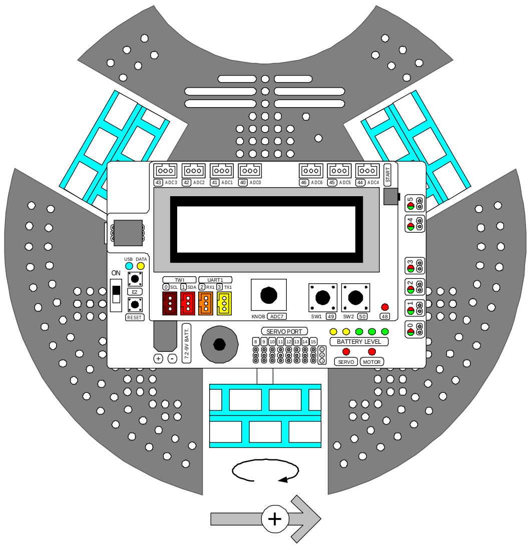
L9.4 Connect the front-left motor to motor output ch. 0 of the Robo-128. Then run the code.

If the connection is correct, Robo-128 will spin anti-clockwise. If not, invert the motor cable connector connection at ch.1 of Robo-128.



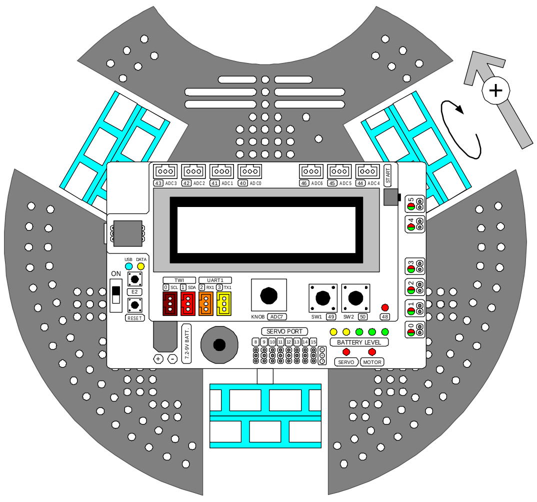
L9.5 Connect the back motor to motor output ch. 1 of the Robo-128.

If the connection is correct, Robo-128 must spin will anti-clockwise. If not, invert the motor cable connector connection at ch.1 of Robo-128.

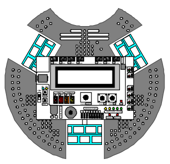


L9.6 Connect the front-right motor to motor output ch. 2 of the Robo-128.

If the connection is correct, Robo-128 will spin anti-clockwise. If not, invert the motor cable connector connection at ch.1 of Robo-128.



After all motor connection are correct, keep these connections for all experiments.



Experiment 10 : Robo-128 movement function

Experiment 10.1 - Move forward function

This experiment demonstrates the forward movement function by Robo-128 after pressing SW1.

L10.1.1 Create a new sketch file and save it as robot_move_forward. Type in the code following the Listing L10-1.

L10.1.2 Compile and upload the sketch file to Robo-128 robot.

L10.1.3Run the code.

Robo-128's LCD shows message

SW1 Press

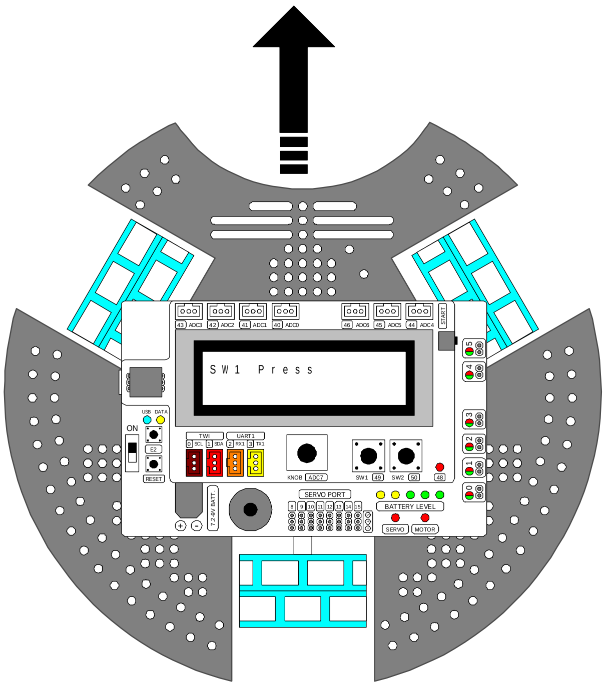
```
#include <robot.h>
void forward(int L,int R)    // Move forward function
{
  motor(0,-L);         // Drives the front-left motor forward by L power
  motor(1,0);          // Stop the back wheel motor
  motor(2,R);          // Drives the front-right motor forward by R power
}
void setup()
{
  lcd("SW1 Press!"); // Display message for pressing SW1
  sw1_press();       // Wait until the SW1 is pressed
}
void loop()
{
  forward(70,70);    // The robot moves forward by 70% of power
}
```

<u>Note</u> : You can change the value of L and R power parameter to the best movement. Possible to set both parameter with different value if the motor is not 100% compatible.

Listing L10-1 : robot_move_forward.pde file; the C/C++ code of Wiring for driving the Robo-128 to move forward

L10.1.4 Press the SW1 and observe the robot’s movement

Robo-128 moves forward after SW1 is pressed.



Experiment 10.2 - Move backward function

This experiment demonstrates the backward movement function operation for Robo-128 after pressing SW1.

L10.2.1 Create a new sketch file and save as robot_move_backward. Type in the code following the Listing L10-2.

L10.2.2 Compile and upload the sketch file to Robo-128 robot.

L10.2.3Run the code.

Robo-128's LCD shows message

SW1 Press

L10.2.4 Press the SW1 and observe the robot's movement

Robo-128 moves backward after SW1 is pressed.

```
#include <robot.h>
void backward(int L,int R)   // Move backward function
{
  motor(0,L);          // Drives the front-left motor backward by L power
  motor(1,0);          // Stop the back wheel motor
  motor(2,-R);         // Drives the front-right motor backward by R power
}
void setup()
{
  lcd("SW1 Press!"); // Display message for pressing SW1
  sw1_press();         // Wait until the SW1 is pressed
}
void loop()
{
  backward(70,70);     // The robot moves backward by 70% of power
}
```

Listing L10-2 : robot_move_backward.pde file; the C/C++ code of Wiring for driving the Robo-128 to move backward

Experiment 10.3 - Move diagonal left forward function

This experiment demonstrates the diagonal left forward movement function's operation for Robo-128 after pressing SW1.

L10.3.1 Create a new sketch file and save as robot_move_front_left. Type in the code following the Listing L10-3.

L10.3.2 Compile and upload the sketch file to Robo-128 robot.

L10.3.3 Run the code.

Robo-128's LCD shows message

```
SW1 Press
```

L10.3.4 Press the SW1 and observe the robot's movement

Robo-128 moves diagonal left forward after SW1 is pressed.

```
#include <robot.h>
voi font_left(int B,int R)   // Move diagonal left forward function
{
  motor(0,0);          // Stop the front-left motor
  motor(1,-B);         // Drives the back wheel motor to left direction
                       // by B power
  motor(2,R);          // Drives the front-right motor forward by R power
}
void setup()
{
  lcd("SW1 Press!"); // Display message for pressing SW1
  sw1_press();         // Wait until the SW1 is pressed
}
void loop()
{
  font_left(80,80);  // The robot move diagonal left forward with
                       // 80% of power
}
```

Listing L10-3 : robot_move_front_left.pde file; the C/C++ code of Wiring for driving the Robo-128 to move diagonal left forward

Experiment 10.4 - Move diagonal left backward function

This experiment demonstrates the diagonal left backward movement function's operation for Robo-128 after pressing SW1.

L10.4.1 Create a new sketch file and save as robot_move_back_left. Type in the code following the Listing L10-4.

L10.4.2 Compile and upload the sketch file to Robo-128 robot.

L10.4.3 Run the code.

Robo-128's LCD shows message

SW1 Press

L10.4.4 Press the SW1 and observe the robot's movement

Robo-128 moves diagonal left backward after SW1 is pressed.

```
#include <robot.h>
voi font_left(int B,int R)   // Move diagonal left backward function
{
  motor(0,0);         // Drives the front-left motor backward by L power
  motor(1,-B);        // Drives the back wheel motor to left direction
                      // by B power
  motor(2,R);         // Stop the front-right motor
}
void setup()
{
  lcd("SW1 Press!"); // Display message for pressing SW1
  sw1_press();        // Wait until the SW1 is pressed
}
void loop()
{
  font_left(80,80);  // The robot move diagonal left backward with
                      // 80% of power
}
```

Listing L10-4 : robot_move_back_left.pde file; the C/C++ code of Wiring for driving the Robo-128 to move diagonal left backward

Experiment 10.5 - Move diagonal right forward function

This experiment demonstrates the diagonal right forward movement function's operation for Robo-128 after pressing SW1.

L10.5.1 Create a new sketch file and save as robot_move_front_right. Type in the code following the Listing L10-5.

L10.5.2 Compile and upload the sketch file to Robo-128 robot.

L10.5.3 Run the code.

Robo-128's LCD shows message

SW1 Press

L10.5.4 Press the SW1 and observe the robot's movement

Robo-128 moves diagonal right forward after SW1 is pressed.

```
#include <robot.h>
voi font_left(int B,int R)   // Move diagonal right forward function
{
  motor(0,0);         // Drives the front-left motor forward by L power
  motor(1,-B);        // Drives the back wheel motor to right direction
                      // by B power
  motor(2,R);         // Stop the front-right motor
}
void setup()
{
  lcd("SW1 Press!"); // Display message for pressing SW1
  sw1_press();        // Wait until the SW1 is pressed
}
void loop()
{
  font_left(80,80);  // The robot move diagonal right forward with
                      // 80% of power
}
```

Listing L10-5 : robot_move_front_right.pde file; the C/C++ code of Wiring for driving the Robo-128 to move diagonal right forward

Experiment 10.6 - Move diagonal right backward function

This experiment demonstrates the diagonal right backward movement function's operation for Robo-128 after pressing SW1.

L10.6.1 Create a new sketch file and save as robot_move_back_right. Type in the code following the Listing L10-6.

L10.6.2 Compile and upload the sketch file to Robo-128 robot.

L10.6.3 Run the code.

Robo-128's LCD shows message

SW1 Press

L10.6.4 Press the SW1 and observe the robot's movement

Robo-128 moves diagonal right backward after SW1 is pressed.

```
#include <robot.h>
voi font_left(int B,int R)   // Move diagonal right backward function
{
  motor(0,0);         // Stop the front-left motor
  motor(1,-B);        // Drives the back wheel motor to right direction
                      // by B power
  motor(2,R);         // Drives the front-right motor backward by R power
}
void setup()
{
  lcd("SW1 Press!"); // Display message for pressing SW1
  sw1_press();        // Wait until the SW1 is pressed
}
void loop()
{
  font_left(80,80);  // The robot move diagonal right backward with
                      // 80% of power
}
```

Listing L10-6 : robot_move_back_right.pde file; the C/C++ code of Wiring for driving the Robo-128 to move diagonal right backward

Experiment 10.7 - Shift left function

This experiment demonstrates the shift left function's operation for Robo-128 after pressing SW1.

L10.7.1 Create a new sketch file and save as robot_move_left. Type in the code following the Listing L10-7.

L10.7.2 Compile and upload the sketch file to Robo-128 robot.

L10.7.3 Run the code.

Robo-128's LCD shows message

```
SW1 Press
```

L10.7.4 Press the SW1 and observe the robot's movement

Robo-128 shifts left after SW1 is pressed.

```
#include <robot.h>
void shift_left(int L,int B,int R)    // Move left function
{
  motor(0,0);          // Drives the front-left motor backward by L power
  motor(1,-B);         // Drives the back wheel motor to left direction
                       // by B power
  motor(2,R);          // Drives the front-right motor forward by R power
}
void setup()
{
  lcd("SW1 Press!"); // Display message for pressing SW1
  sw1_press();         // Wait until the SW1 is pressed
}
void loop()
{
  font_left(80,80);  // The robot move left by
                       // 35% of power of the front-left motor,
                       // 80% of power of the back motor and
                       // 35% of power of the front-right motor
}
```

Listing L10-7 : robot_move_left.pde file; the C/C++ code of Wiring for driving the Robo-128 to move (or shift) left

Experiment 10.8 - Shift right function

This experiment demonstrates the shift right function's operation for Robo-128 after pressing SW1.

L10.8.1 Create a new sketch file and save as robot_move_right. Type in the code following the Listing L10-7.

L10.8.2 Compile and upload the sketch file to Robo-128 robot.

L10.8.3 Run the code.

Robo-128's LCD shows message

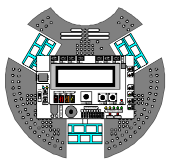
SW1 Press

L10.8.4 Press the SW1 and observe the robot's movement

Robo-128 shifts right after SW1 is pressed.

```
#include <robot.h>
void shift_right(int L,int B,int R)       // Move right function
{
  motor(0,0);         // Drives the front-left motor forward by L power
  motor(1,-B);        // Drives the back wheel motor to right direction
                      // by B power
  motor(2,R);         // Drives the front-right motor backward by R power
}
void setup()
{
  lcd("SW1 Press!"); // Display message for pressing SW1
  sw1_press();        // Wait until the SW1 is pressed
}
void loop()
{
  font_left(80,80);   // The robot move right by
                      // 35% of power of the front-left motor,
                      // 80% of power of the back motor and
                      // 35% of power of the front-right motor
}
```

Listing L10-8 : robot_move_right.pde file; the C/C++ code of Wiring for driving the Robo-128 to move (or shift) right



Experiment 11 : Robo-128 with the infrared ball

This experiment presents easy examples of C/C++ programming to control the Robo-128 to detect an infrared ball. There is some condition as follows :

1. Detection of the direction of the ball to orientate the robot to faces the ball.

2. When the infrared ball is in front of the robot and near to the robot enough, Robo-128 generates a sound signifying ball detection.

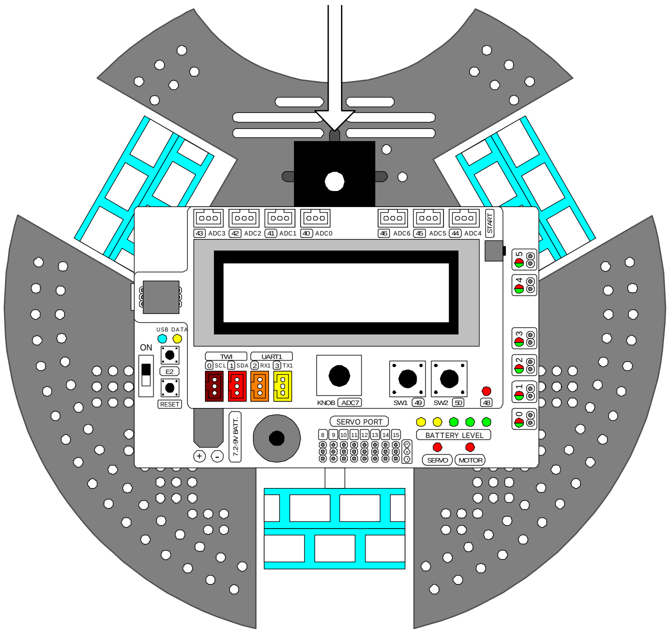
Connecting additional hardware

l Connect the Direction output of ZX-IrEYE to ADC0 port of Robo-128.

l Connect the Range output of ZXX-IrEYE to ADC1 port of Robo-128.

l Attach the ZX-IrEYE at the front of the Robo-128 to secure and turn the componenet side down to the chasis following the picture below.

Attach the ZX-IrEYE at front of the robot and
turn the component side down to the chasis.



Procedure

L11.1 Create a new sketch file and save it as robot_detect_ball. Type in the code following the Listing L111-1

L11.2 Compile and upload the sketch file to Robo-128 robot.

```
#include <robot.h>
int dir;                    // Declare the Direction variable
int range;                  // Declare the Range variable
void spin_left(int p)       // Spin left function
{
   motor(0,p);              // Drives the front-left motor backward by p power
   motor(1,p);              // Drives the back wheel motor to right direction by p power
   motor(2,p);              // Drives the front-right motor forward by p power
}
void spin_right(int p)      // Spin right function
{
   motor(0,-p);             // Drives the front-left motor forward by p power
   motor(1,-p);             // Drives the back wheel motor to left direction by p power
   motor(2,-p);             // Drives the front-right motor backward by p power
}
void setup()
{
   lcd("SW1 Press!");       // Display message for pressing SW1
   sw1_press();             // Wait until the SW1 is pressed
}
void loop()
{
   dir = analog(0);         // Get the direction from sensor
   range = analog(1);       // Get the range from sensor
   lcd("DIR: %d    #nRANGE: %d    ",dir,range);
                            // Displays the direction value on upper line of LCD
                            // and the range value on lower line of LCD
   if(dir<400)              // Check the ball at right-side of the robot
   {
     spin_right(80);        // If yes, spin right to meet the ball
   }
   else if(dir>=400 && dir<=500)// Check the ball at front-side of the robot
   {
     motor_stop(ALL);       // Robot does not move if the ball is in front of the robot
     if(range>1000)         // Check the ball very near the robot

     {
        beep();             // Drive sound if the ball is very near the robot
     }
   }
   else                     // The ball is left-side of the robot
   {
     spin_left(80);         // Spin left to meet the ball
   }
}
```

Code description

Program begins with the LCD module to display text SW1 Press! to wait the switch SW1 pressing. After SW1 is pressed, program works within the loop function to read values from the ZX-IrEYE sensor. They include direction and range of the infrared ball. After that, microcontroller analyses these values to tries to adjust the front of robot correspond the infrared ball position.

The reference direction value could be change. In this code, select 400 to 500 as the position at robot get the ball. Next, robot is stop and check distance or range from robot to the infrared ball. If the ball is in scope (range = 1000 or more), beep at once.

Listing L11-1 : robot_detect_ball.pde file; the C/C++ code of Wiring for testing the infrared ball detecion of the Robo-128

L11.3 Run the code.

Robo-128's LCD shows the message

SW1 Press

L11.4 Press the SW1 and observe the robot's movement

Robo-128's LCD shows message

DIR: xxx

RANGE: yyy

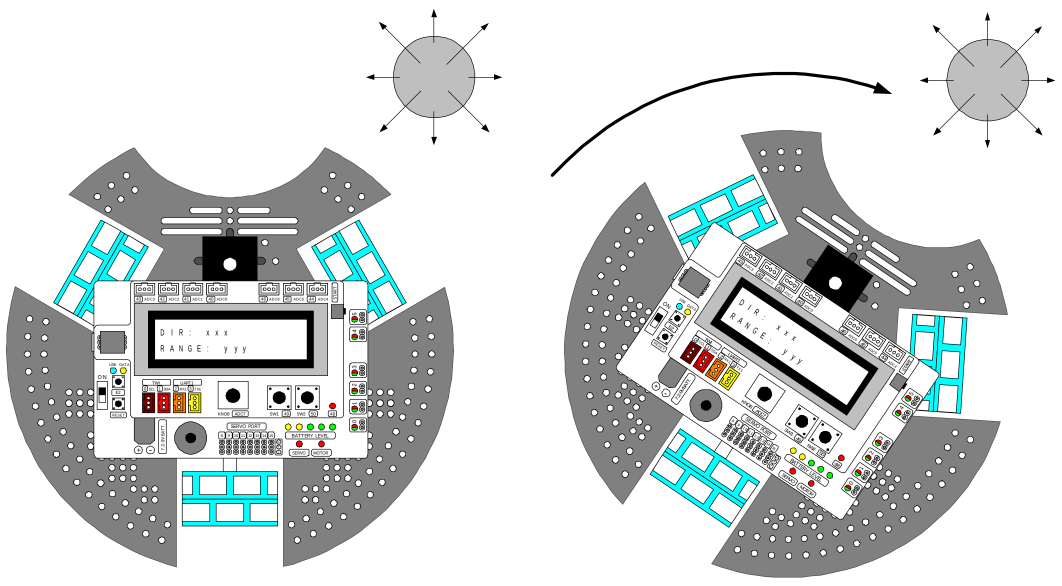
Therefore

xxx *is the direction tendency value of the infrared source. In this case is the infrared ball. The value is between 0 to 1023. In case attach the sensor with make face down, the sensor gives high value if the infrared ball locate at left side of sensor. It gives low value if the infrared ball locate at right side of sensor.*

yyy *is the distance value between the infrared source and sensor. The value is 0 to 1023. The sensor gives high value when near the infrared source and decrease if the infrared source is far away.*

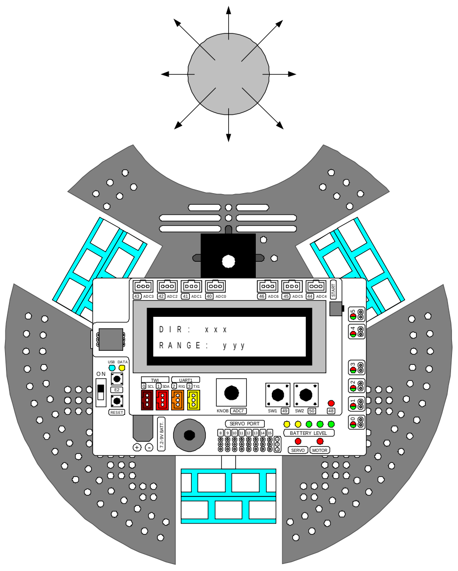
L11.5 Place the infrared ball at front-right of the Robo-128 about 10cm. Observe the robot's operation.

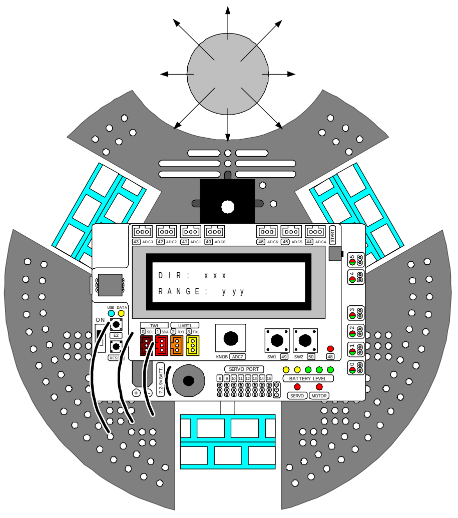
Robo-128 robot responds with rotate right to face the ball.

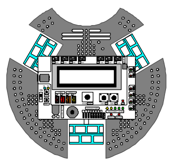


L11.6 Place the infrared ball in front of the Robo-128 about 30cm. Shift the ball to the robot slowly. Observe the robot's operation.

Robo-128 responses by stop the spinning and check the distance from the infrared ball. When the Range value is more than 1000 (see at LCD screen of the robot), Robo-128 drives a beep to inform the ball detection.





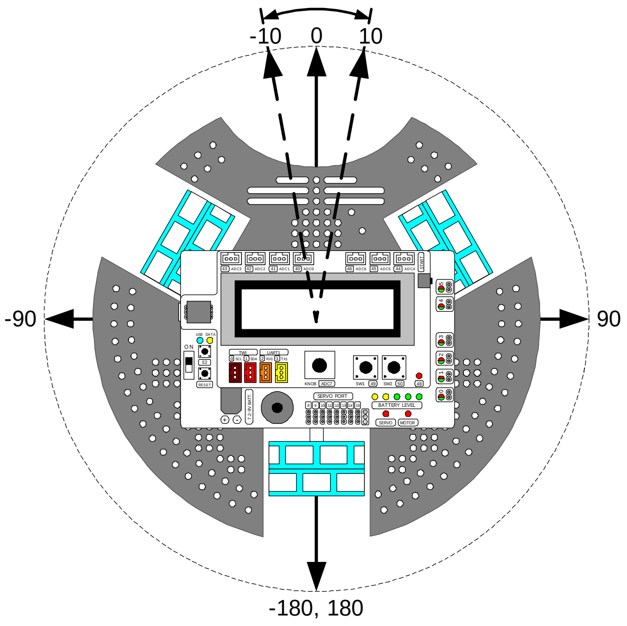


Experiment 12 : Robo-128 navigation by using the digital compass

This experiment demonstrates the simple robot navigation by using the C/C++ programming to control the Robo-128 to work with a digital compass an infrared ball. There is some condition as follows :

1. Robot must try to move in the direction of the reference angle between -10 to 10 degrees.

2. When the robot is out of bounds, it must be spin back to the scope of the reference angle between -10 to 10 degrees following the picture below.



L12.1 Create a new sketch file and save as robot_compass_direction. Type in the code following the Listing L12-1.

L12.2 Compile and upload the sketch file to Robo-128 robot.

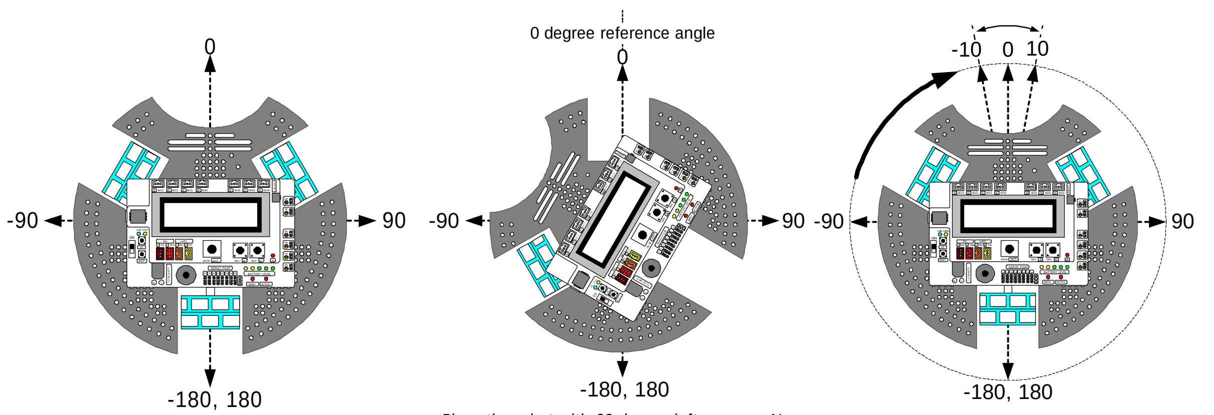
L12.3 Run the code.

Robo-128's LCD shows message

```
SW1 Press
```

L12.4 Press the SW1 and observe the robot's movement.

Robo-128 will begin to move forward as usual. Since all is restarted, the reference angle 0 degree is the front angle of robot.



Place the robot. Press START switch. The robot gets value from the digital comapss and set the front direction as the 0 degree reference angle.

Place the robot with 60 degree left or more. No need to press the START to restart.

Release your hand from the robot. It rotates to set its postion in range -10 to +10 degree from the reference angle.

```
#include <robot.h>
#include <compass.h>      // Include the digital compass library
int angle;                // Declare the Angle variable
void spin_left(int p)     // Spin left function
{
   motor(0,p);            // Drives the front-left motor backward by p power
   motor(1,p);            // Drives the back wheel motor to right direction by
p power
   motor(2,p);            // Drives the front-right motor forward by p power
}
void spin_right(int p)    // Spin right function
{
   motor(0,-p);           // Drives the front-left motor forward by p power
   motor(1,-p);           // Drives the back wheel motor to left direction by
p power
   motor(2,-p);           // Drives the front-right motor backward by p power
}
void forward(int L,int R) // Move forward function
{
   motor(0,-L);           // Drives the front-left motor forward by L power
   motor(1,0);            // Stop the back wheel motor
   motor(2,R);            // Drives the front-right motor forward by R power
}
void setup()
{
   lcd(“SW1 Press!”);     // Display message for pressing SW1
   sw1_press();           // Wait until the SW1 is pressed
   compass_set_heading(); // Set heading angle reference
}
void loop()
{
   angle = compass_read_heading();
                          // Read thew angle value in -180 to 180 degree format
                          // from heading refrence angle
```

Listing L12-1 : robot_compass_direction.pde file; the C/C++ code of Wiring for Robo-128 navigation by using the digital compass sensor (continue)

```
   if(angle>=-180 && angle<-10)   // Check the robot is slighting left
   {
     spin_right(50);              // If yes, spin right to adjust the position
                                  // of robot into the angle range is -10 to 10
                                  // degree from heading reference angle
   }
   else if(angle>=-10 && angle<=10)
                                  // Check the robot position within the hading
                                  // angle range ( -10 to 10 degree)
   {
     forward(80,80);              // If yes, move forward
   }
   else                           // If the robot is slighting right
   {
     spin_left(50);               // Spin left to adjust the position of robot
                                  // into the angle range is -10 to 10 degree
                                  // from heading reference angle
   }
}
```

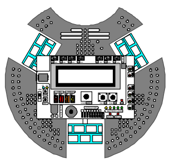
Code description

Program begins with the LCD module to display text SW1 Press! to wait the switch SW1 pressing. After SW1 is pressed, program set the new heading angle with the current angle that is get from the digital compass. After that works within the loop function to read values from the compass sensor in -180 to 180 degree format. Robot will use this value to control its position within range -10 to 10 degree from the heading angle.

Listing L12-1 : robot_compass_direction.pde file; the C/C++ code of Wiring for Robo-128 navigation by using the digital compass sensor (continue)

L12.5 Turn the robot to another direction. Observe the robo's operation without restart.

The Robo-128 must try to move in the direction of the reference angle between -10 to 10 degrees.



Chapter 7
Remote color sensing of Robo-128

Robo-128 is capable enough to be use in the modern World's junior soccer robot competition which defines the goal of each side with 2 different colors; yellow and blue. The remote color sensor is an important sensor to the robot. The sensor's operation is like a camera.

ZX-CCD is the suggested sensor for Robo-128 for this application. This chapter describes the ZX-CCD operation and some examples of C/C++ programming to work with Robo-128 for color detection.

7.1 Introduction to ZX-CCD Vision board

7.1.1 ZX-CCD features

- Track user defined color blobs at 17 Frames Per Second
- Find the centroid of the blob
- Gather mean color and variance data
- Transfer a real-time binary bitmap of the tracked pixels in an image
- Arbitrary image windowing
- Adjust the camera's image properties
- Dump a raw image
- 80x143 Resolution
- 115,200 / 38,400 / 19,200 / 9600 baud serial communication
- Slave parallel image processing mode off a single camera bus

7.1.2 Technical introduction

The ZX-CCD can be used to track or monitor color. The best performance can be achieved when there are highly contrasting and intense colors. For instance, it could easily track a red ball on a white background, but it would be hard to differentiate between different shades of brown in changing light.

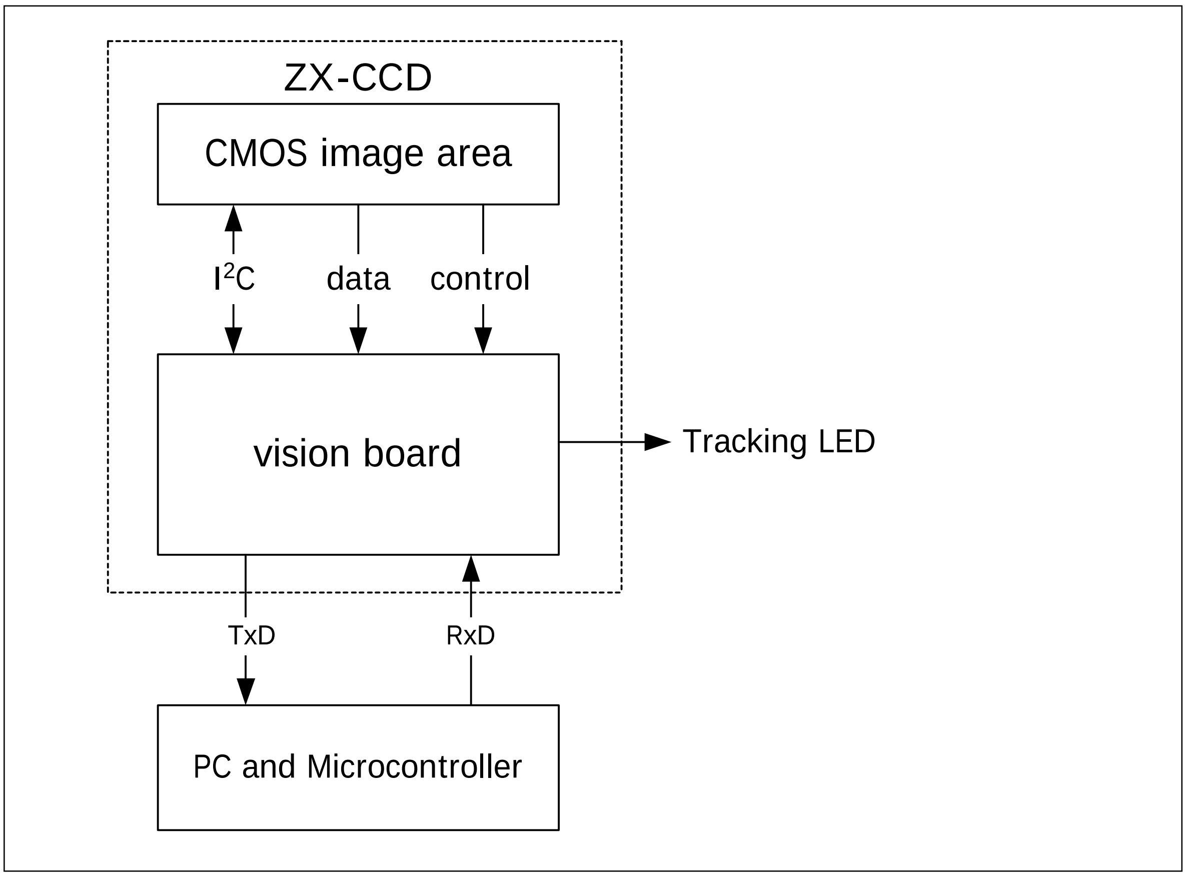
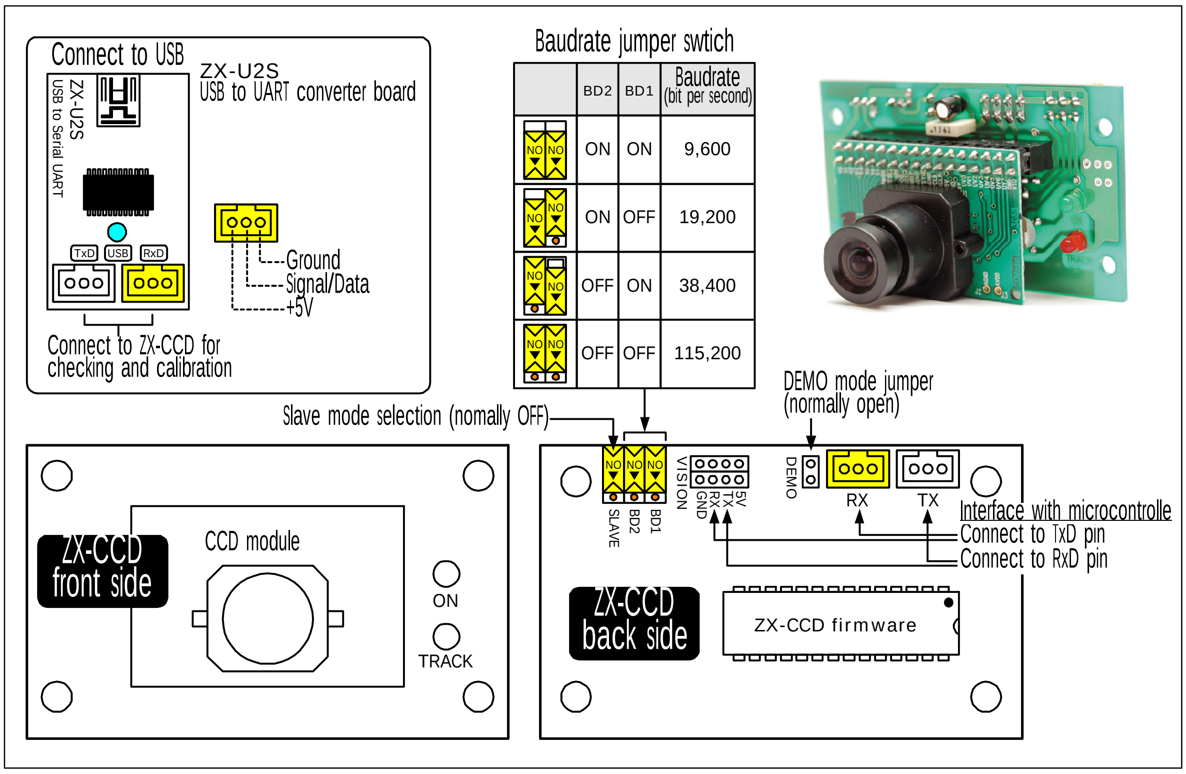


Figure 7-1 ZX-CCD diagram

Tracking colorful objects can be used to localize landmarks, follow lines, or chase a moving beacon. Using color statistics, it is possible to monitor a scene, detect a specific color or do primitive motion detection. If the camera detects a drastic color change, then chances are something in the scene changed. Using "line mode", the ZX-CCD can act as an easy way to get low resolution binary images of colorful objects. This can be used to do more sophisticated line following that includes branch detection, or even simple shape recognition. These more advanced operations would require custom algorithms that would post process the binary images sent from the ZX-CCD. As is the case with a normal digital camera, this type of processing might require a computer or at least a fast microcontroller.

The most common configuration for the ZX-CCD is to have it communicate to a computer via USB port by using ZX-U2S USB to serial converter board (included in package) The ZX-CCD is small enough to add simple vision to embedded systems that can not afford the size or power of a standard computer based vision system. Its communication protocol is designed to accommodate even the slowest of processors. If your device does not have a fully level shifted serial port, you can also communicate to the ZX-CCD over the TTL serial port. This is the same as a normal serial port except that the data is transmitted using 0 to 5 volt logic.



	BD2	BD1	Baudrate (bit per second)
	ON	ON	9,600
	ON	OFF	19,200
	OFF	ON	38,400
	OFF	OFF	115,200

Figure 7-2 The ZX-CCD hardware configuration

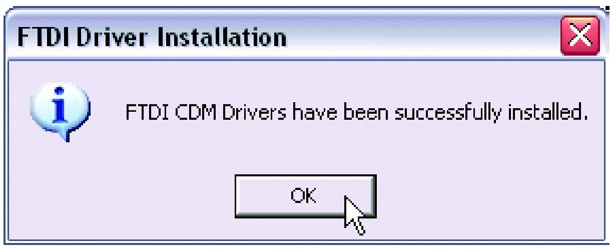
The ZX-CCD supports various baud rates to accommodate slower processors. For even slower processors, the camera can operate in “poll mode”. In this mode, the host processor can ask the CMUcam for just a single packet of data. This gives slower processors the ability to more easily stay synchronized with the data. It is also possible to add a delay between individual serial data characters using the “delay mode” command. Due to the communication delays, both poll mode and delay mode will lower the total frame rate that can be processed.

7.1.3 Supply Preparation

The suitable power supply for ZX-CCD is +5V. User can use from both signal connector RxD or TxD from microcontroller board or ZX-U2S (in calibration mode). The ZX-CCD need 70mA current consumption.

7.2 ZX-U2S Driver installation

To test the ZX-CCD, you need a computer. ZX-CCD can interface computer via USB pot by using the ZX-U2S board. First, you must install the USB driver. Double click at USBDriverInstallerV2.x.x.exe (x.x is number of version) file from the bundled CD-ROM to start the driver installation. The installation dialogue-box will appear below.



FTDI Driver Installation

FTDI CDM Drivers have been successfully installed.

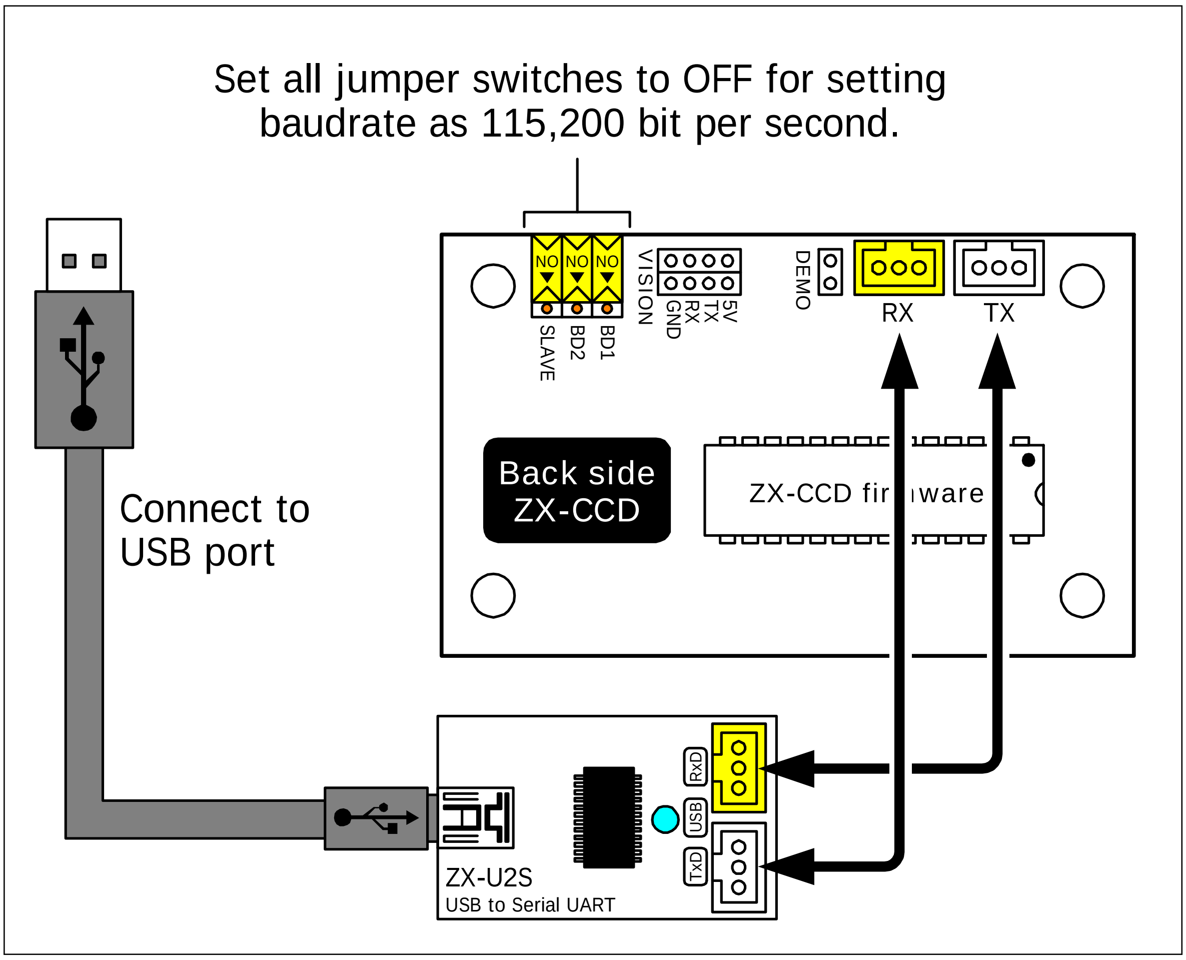
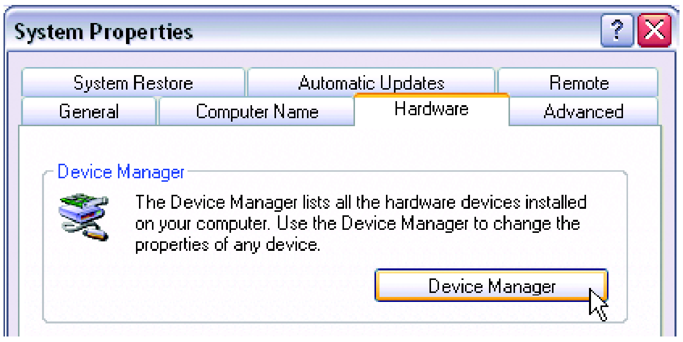


Figure 7-3 ZX-CCD with computer connection diagram

7.2.1 Check the USB serial port address

(1) Plug the USB cable or ZX-U2S. Wait until the LED indicator at USB position on ZX-U2S is turned-on.

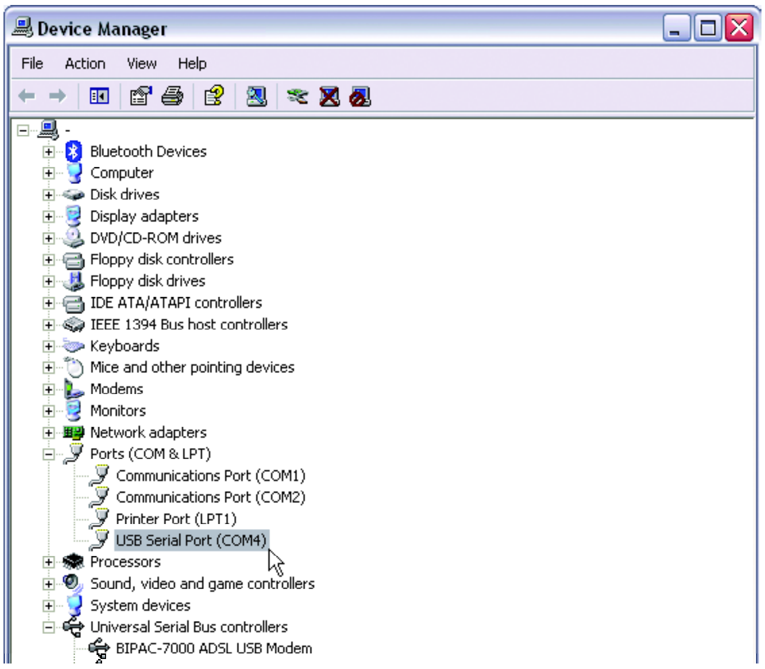
(2) Check the Virtual COM port or USB Serial port address by clicking Start à Control Panel à System à Hardware à Device Manager



System Properties

The Device Manager lists all the hardware devices installed on your computer. Use the Device Manager to change the properties of any device.

(3) See the list of USB serial port and remember the COM port address to work with ZX-U2S board. Normally it will create COM3 or higher. In this example is COM4



Bluetooth Devices
Computer
Disk drives
Display adapters
DVD/CD-ROM drives
Floppy disk controllers
Floppy disk drives
IDE ATA/ATAPI controllers
IEEE 1394 Bus host controllers
Keyboards
Mice and other pointing devices
Modems
Monitors
Network adapters
Ports (COM & LPT)
Communications Port (COM1)
Communications Port (COM2)
Printer Port (LPT1)
USB Serial Port (COM4)
Processors
Sound, video and game controllers
System devices
Universal Serial Bus controllers
BIPAC-7000 ADSL USB Modem

7.2.2 Virtual COM port with CMUCAM2GUI and RS-232 Terminal operation notice

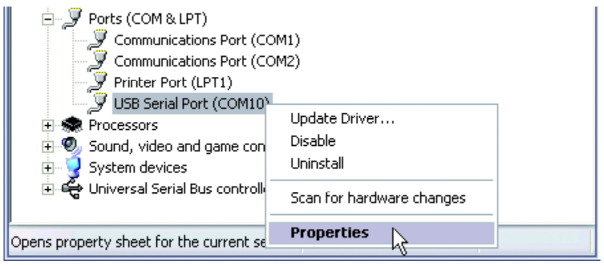
Normally CMUCAM2 and RS-232 Terminal software can interface with COM port not higher than COM9. Thus, you must make sure the USB serial port address not higher than COM9. If its higher, please do following procedures.

(1) Connect the ZX-U2S to Computer USB port.

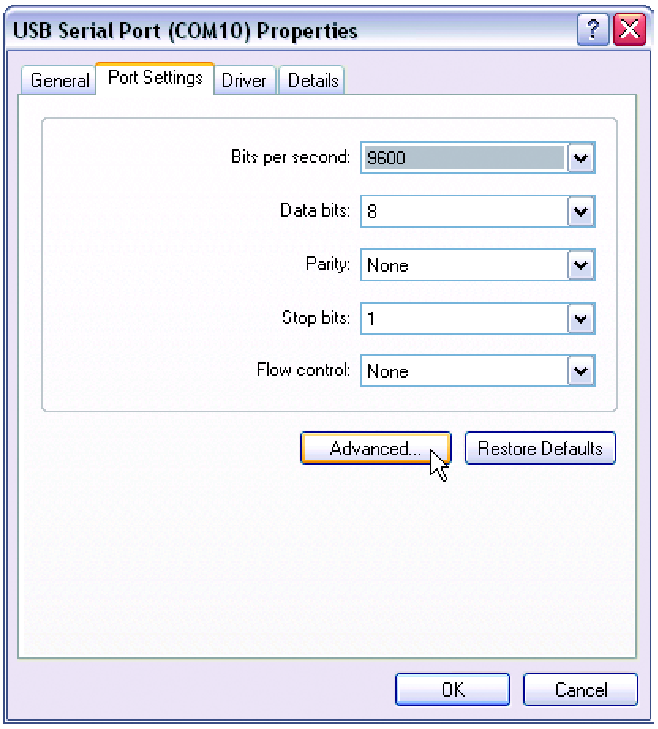
(2) Check the COM port address by clicking at Start à Control Panel à System

(3) Select the Hardware tab and click on the Device Manager button.

(4) Check the hardware listing. At the Port listing, you will found USB Serial port (COM x).If COM port is higher than COM9 (this example is COM10), please click on the right-button mouse and select to Properties

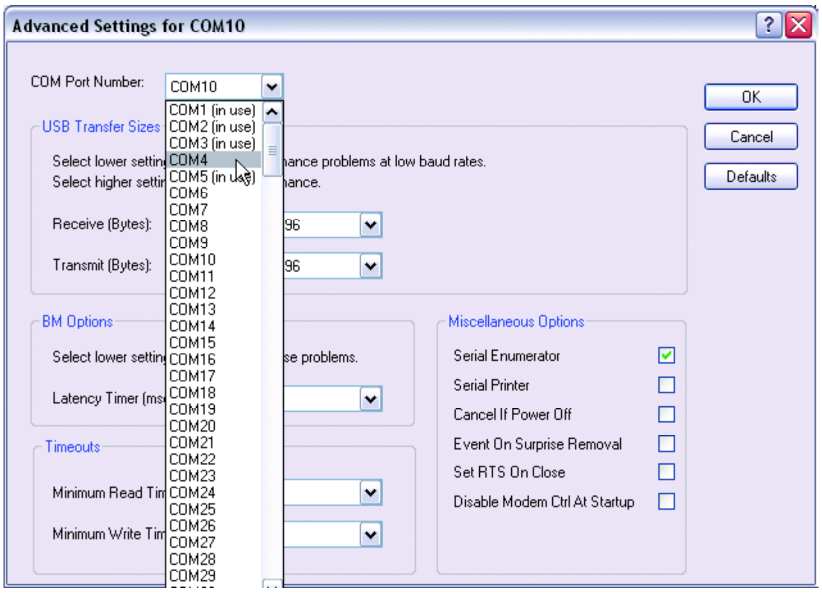

Ports (COM & LPT)
Communications Port (COM1)
Communications Port (COM2)
Printer Port (LPT1)
USB Serial Port (COM10)
Processors
Sound, video and game con
System devices
Universal Serial Bus controll
Update Driver...
Disable
Uninstall
Scan for hardware changes
Properties
Opens property sheet for the current se

(5) The USB Serial Port (COM10) Properties window will appear. Select the Port Setting tab and set all value following the figure below and click on the Advance button



Bits per second: 9600
Data bits: 8
Parity: None
Stop bits: 1
Flow control: None

(6) The Advanced Setting for COM10 will appear. Click on the COM Port Number box to change to *COM4* or another port in range *COM1* to *COM9*.



Advanced Settings for COM10

COM Port Number: COM10

COM1 (in use)
COM2 (in use)
COM3 (in use)
COM4
COM5 (in use)
COM6
COM7
COM8
COM9
COM10
COM11
COM12
COM13
COM14
COM15
COM16
COM17
COM18
COM19
COM20
COM21
COM22
COM23
COM24
COM25
COM26
COM27
COM28
COM29

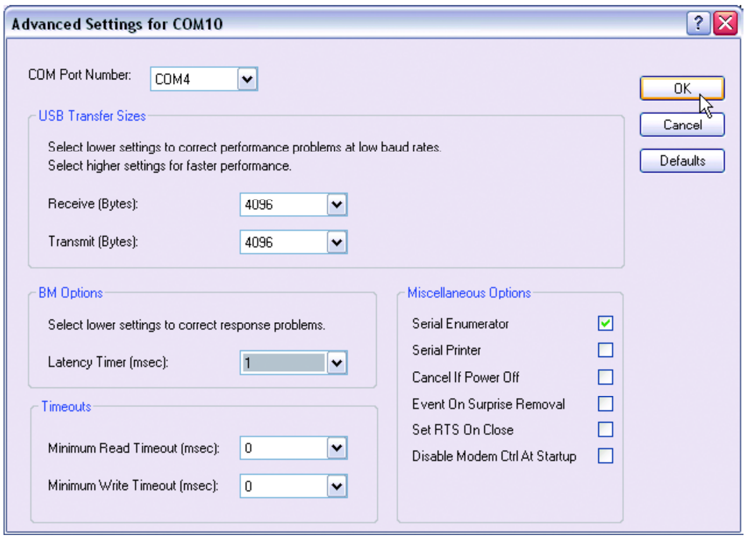
USB Transfer Sizes
Select lower settin… …ance problems at low baud rates.
Select higher settir… …ance.
Receive (Bytes): …96
Transmit (Bytes): …96

BM Options
Select lower settin… …se problems.
Latency Timer (ms…

Timeouts
Minimum Read Ti…
Minimum Write Ti…

Miscellaneous Options
Serial Enumerator ☑
Serial Printer ☐
Cancel If Power Off ☐
Event On Surprise Removal ☐
Set RTS On Close ☐
Disable Modem Ctrl At Startup ☐

(7) Set the value following the figure below. Especially at the Latency Timer (msec) suggested to set to 1 and check the box at Serial Enumerator. Click OK button.



Advanced Settings for COM10

COM Port Number: COM4

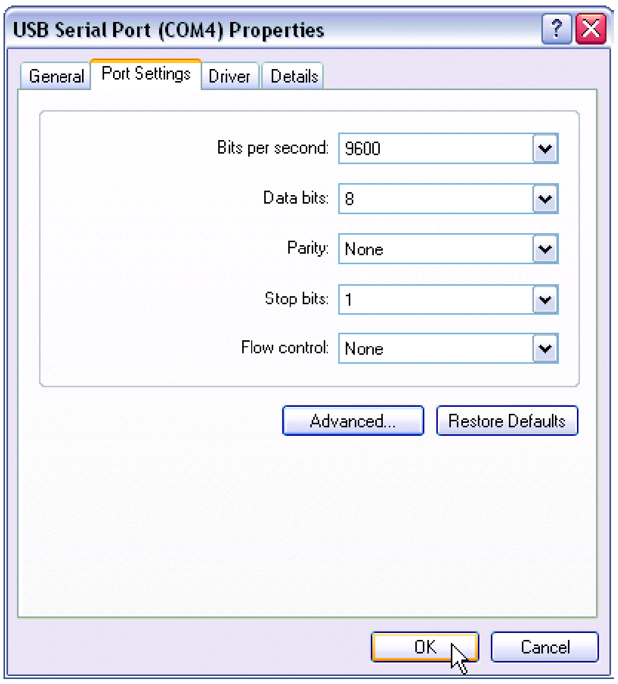
USB Transfer Sizes
Select lower settings to correct performance problems at low baud rates.
Select higher settings for faster performance.
Receive (Bytes): 4096
Transmit (Bytes): 4096

BM Options
Select lower settings to correct response problems.
Latency Timer (msec): 1

Timeouts
Minimum Read Timeout (msec): 0
Minimum Write Timeout (msec): 0

Miscellaneous Options
Serial Enumerator ☑
Serial Printer ☐
Cancel If Power Off ☐
Event On Surprise Removal ☐
Set RTS On Close ☐
Disable Modem Ctrl At Startup ☐

(8) Go back to the USB Serial Port Properties. Now the *COM port number at the title bar will change to COM4. Click on the* OK button.



Bits per second: 9600
Data bits: 8
Parity: None
Stop bits: 1
Flow control: None

(9) Remove the ZX-U2S from USB port and re-plug again. Check the USB Serial port address. The new address must be COM4. Now the ZX-U2S is ready for using with all ZX-CCD software.

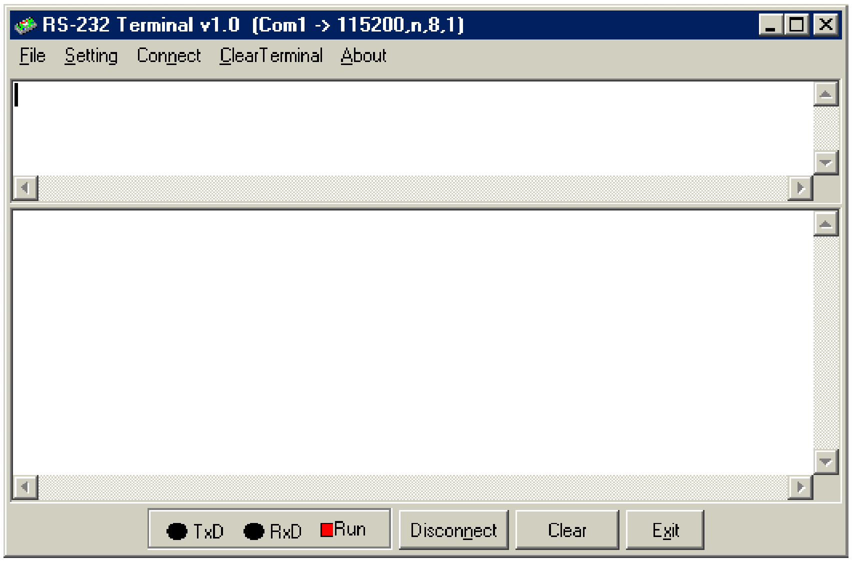
7.3 Testing ZX-CCD

The easiest way to test the ZX-CCD's operation is do computer testing. Two softwares would be used in this tesing; RS-232 Terminal and CMUCAM2GUI. The RS-232 Terminal is used for interfacing, reading and control. CMUCAM2GUI is used for focus adjustment.

Step of testing as :

(1) Install RS-232 Terminal software by click RS-232 TerminalSetup.exe file in CD-ROM and click OK until install complete.

(2) Run by select Start à Program à RS-232 Terminal. Main window of this program will appear.


RS-232 Terminal v1.0 (Com1 -> 115200,n,8,1)
File Setting Connect ClearTerminal About
TxD RxD Run
Disconnect
Clear
Exit

(3) Enter menu Setting à Serial port to select serail port connected. After that select baudrate at same menu; Setting à Baud rate (bps) to 115200, select data bit by Setting à Data to 8-bit and select parity bit by menu Setting à Parity to None. Click Connect Button to connect ZX-CCD with serial port. User can adjust the text format by entered to menu Setting à Terminal

(4) Remove BD1 and BD2 jumper for select baudrate 115,200 bits per second. Remove DEMO and SLAVE jumper too.

(5) Apply supply voltage to ZX-CCD. At output window of RS-232 Terminal will show message :

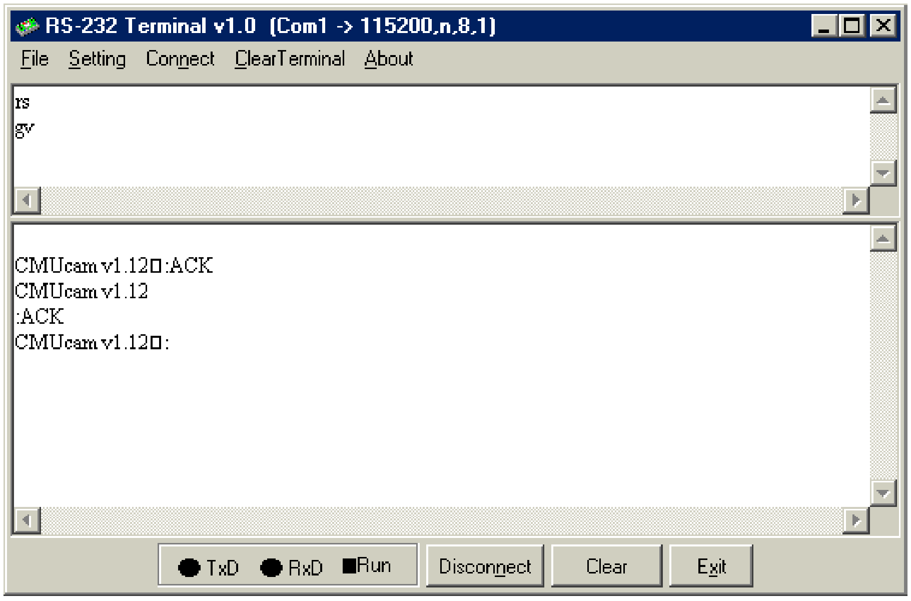
```
CMUcam V1.12 :
```

(6) Type RS command into input window of RS-232 Terminal, press Enter for testing reset ZX-CCD. This message will show :

```
:ACK
CMUCAM1.12
```

(7) Type GV command for reading version number of firmware. This message will show :

```
:ACK
CMUCAM1.12
```


RS-232 Terminal v1.0 (Com1 -> 115200,n,8,1)
File Setting Connect ClearTerminal About
TxD RxD Run
Disconnect
Clear
Exit

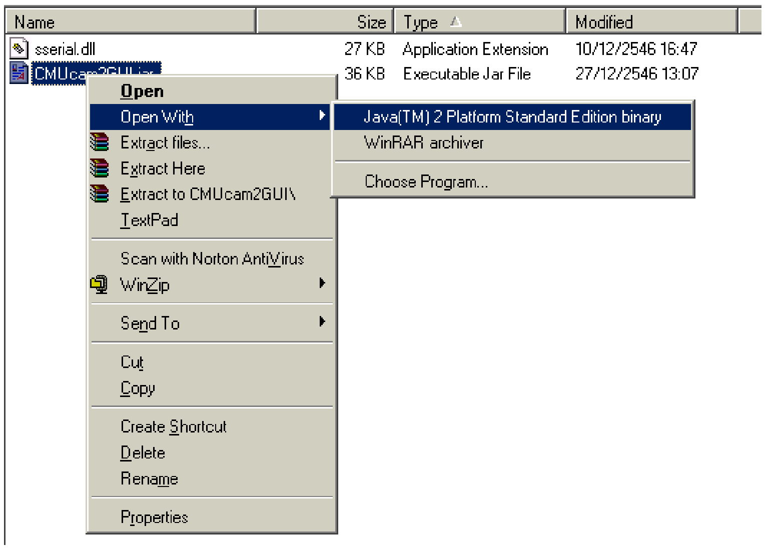
```
rs
gv
```

```
CMUcam v1.12□:ACK
CMUcam v1.12
:ACK
CMUcam v1.12□:
```

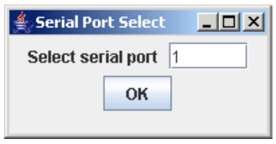
(8) If the ZX-CCD works according to the above, it confirms that ZX-CCD is ready to interface with your computer. Next testing is focusing. The software testing is CMUcam2GUI. Copy file *CMUcam2GUI.jar* that contain in *CMUcam2GUI* folder in CD-ROM into user's harddisk.

(9) CMUcam2GUI required JaVA runtime to work. You need to install the JAVA runtime machine in your system. If you do not have this, install this software from ZX-CCD CD-ROM by running the *jre-1_5_0_04-windows-i586-p.exe* file in the CD-ROM . Click OK until installation completes.

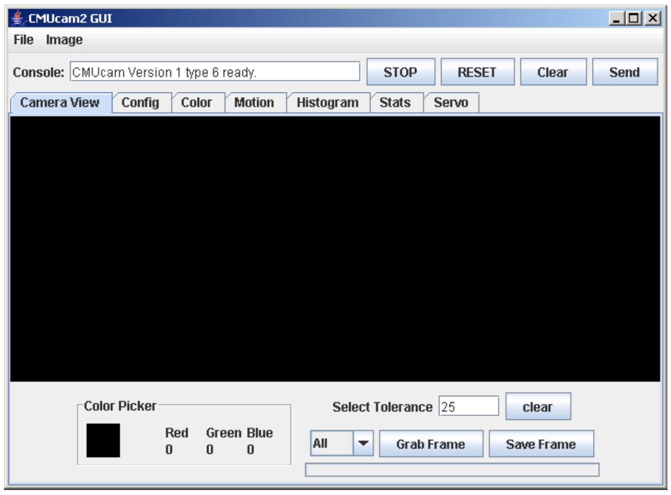
(10) Enter to folder CMUcamGUI **à** Stand_alone. Open file *CMUcam2GUI.jar* by right-button mouse and select Java(TM)2 Platform Standard Edition Binary. See the figure below



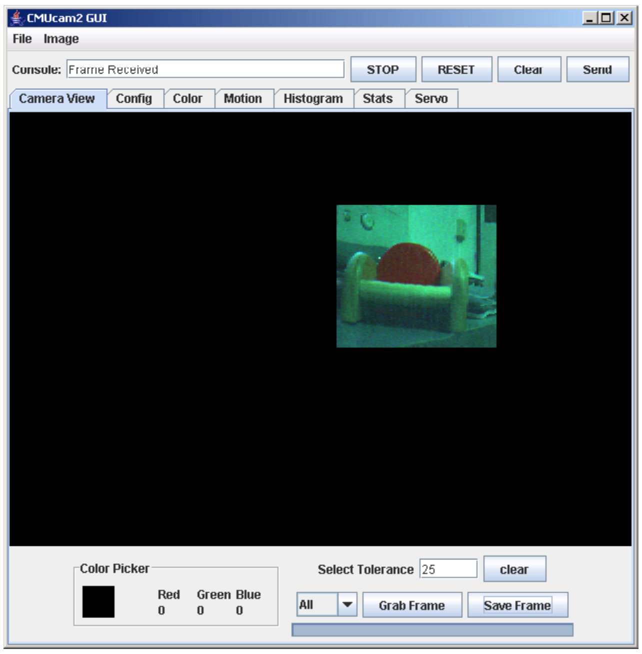
(11) Serial port selection window will appear.



(12) Select the serial port that is created by Virtual COM port driver in Topic 3 of this documentation. The main window of this software will appear.Console box show message : CMUcam Version 1 type 6 ready. It means ZX-CCD connect with CAMcam2GUI successful.


Console: CMUcam Version 1 type 6 ready.

(13) Remove cover rubber from the camera lens. Click the Grab Frame button. Wait a moment. The image that grab will display on left screen of main window. But the image will be mirrored. Enter to menu Image select Flip Horizontal. The image will filp to normal.



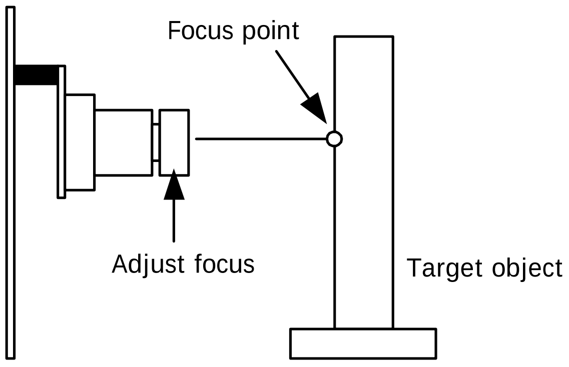
Console: Frame Received

Select Tolerance 25

All

Red	Green	Blue
0	0	0

(14) Once user have the ability to grab frames from the camera, user should be able to rotate the front part of the ZX-CCD lens and see the image change. Try to get the picture to be as sharp as possible by dumping frames and changing the position of the lens a small amount each time. Usually the camera is in focus when the lens is a few rotations away from the base. Adjust until the image grabbed sharp and clear.



7.4 Robo-128 with ZX-CCD interfacing

The figure 7-4 shows the interfacing diagram of Robo-128 with ZX-CCD module. It requires 2 signal cables to connect between Tx port of Robo-128 with Rx pin of the ZX-CCD module and Rx port of Robo-128 with Tx pin of the ZX-CCD module. The interface cable supports +5V supply voltage and ground already. Thus, not requires more power linefor this interfacing.

Installation of the ZX-CCD module should not exceed the chasis of the Robo-128 and above the infrared ball height. There is 2 important reason as follows :

1. Reduce the chance of sensor can be damaged by a collision between robots and impact with the infrared ball in the case of the soccer robot competition.

2. Size of the robot does not increase after installing the ZX-CCD modules.

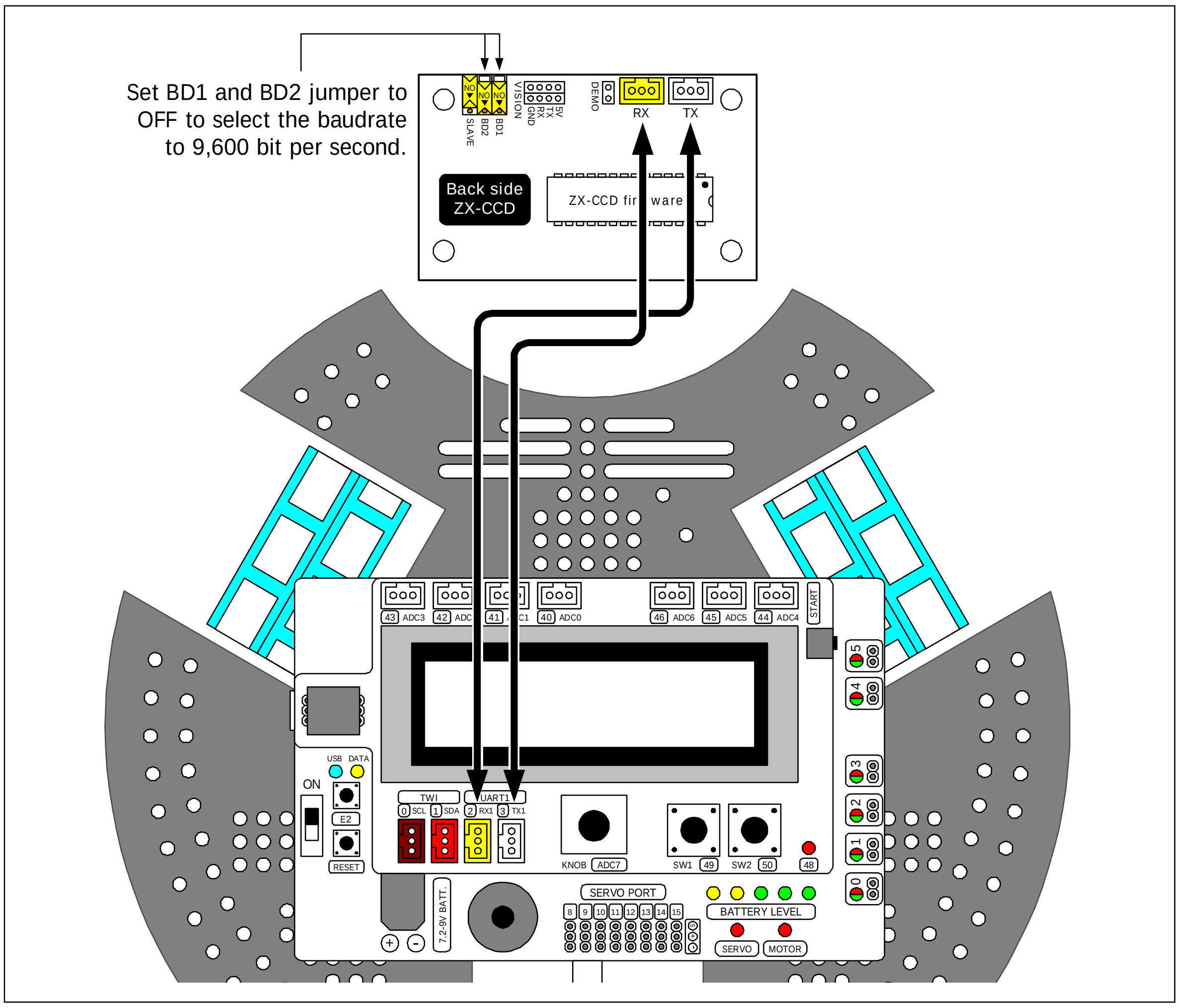


Figure 7-4 The connection diagram of Robo-128 with ZX-CCD module

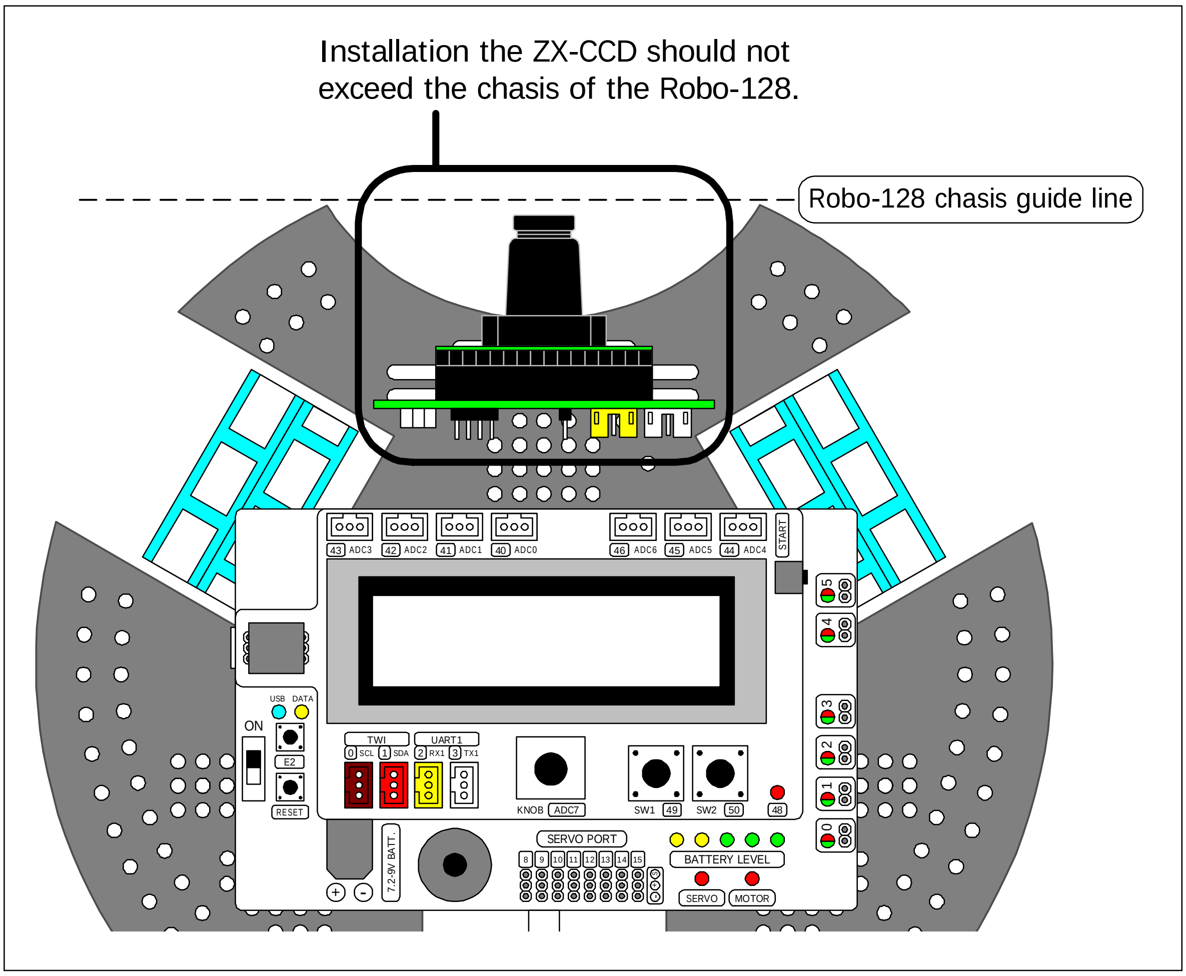


Figure 7-5 Example of installation the ZX-CCD module to Robo-128

7.5 ZX-CCD library file : cam.h

This is the library file that contains instructions about how to interface the ZX-CCD module for analyzing and processing images. You need to include this library file at the top of code by :

```
#include <cam.h>
```

The important functions of this library are as follows :

7.5.1 cam_init

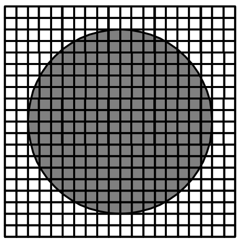
This is the initialize function of the ZX-CCD. The developer must run first before working with the ZX-CCD by using the other functions. This function takes to run about 0.4 second.

Syntax

void cam_init()

7.2.2 cam_get_mean

This means getting the value of the target object function in RGB format.



This function sets the ZX-CCD to detect the object in front. analyze and correct data of each pixel to calculate the mean value of this object.

Syntax

unsigned char * cam_get_mean()

Return value

Address of 8-byte data that contains the RGB mean value. It inlcudes :

1st byte : mean of red composition

2nd byte : mean of green composition

3rd byte : mean of blue composition

4th byte : mean of deviation of red composition

5th byte : mean of deviation of green composition

6th byte : mean of deviation of blue composition

Code example is as follows.

```
unsigned char * m;  // Declare pointer to read data
......................
m = cam_get_mean();
```

Developers can read value from m[0], m[1], m[2], m[3], m[4] and m[5] to convert to array data type. Therefore :

m[0] contains the mean of red composition. It is 0 to 255.

m[1] contains the mean of green composition. It is 0 to 255.

m[2] contains the mean of blue composition. It is 0 to 255.

m[3] contains the mean of deviation of red composition

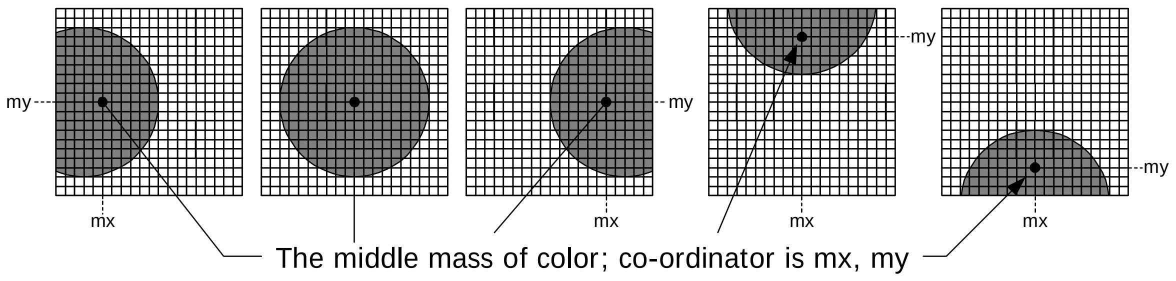
m[4] contains the mean of deviation of green composition

m[5] contains the mean of deviation of blue composition

Note : Almost get only m[0], m[1] and m[2] to track the target object's color with the cam_track_color function.

7.2.3 cam_track_color

This is the color tracking function. The resolution of the frame is 80 x 143 (middle x axis as 40 and middle y axis as 71)



Syntax

unsigned char * cam_track_color (unsigned char rmin, unsigned char rmax,
unsigned char gmin,unsigned char gmax,
unsigned char bmin,unsigned char bmax)

Parameter

rmin - Minimum value of the mean of red composition of target object (0 to 255)

rmax - Maximum value of the mean of red composition of target object (0 to 255)

gmin - Manimum value of the mean of green composition of target object (0 to 255)

gmax - Maximum value of the mean of green composition of target object (0 to 255)

bmin - Minimum value of the mean of blue composition of target object (0 to 255)

bmax - Maximum value of the mean of blue composition of target object (0 to 255)

Return value

Address of 8-byte data that contains the RGB meann value. It inlcudes :

1st byte : x position of middle mass

2nd byte : y position of middle mass

3rd byte : x1 - the left edge position in x axis

4th byte : y1 - the left edge position in y axis

5th byte : x2 - the right edge position in x axis

6th byte : y2 - the right edge position in y axis

7th byte : number of pixel in track region

8th byte : confidence value; maximum is 255

Code example is as follows.

```
    unsigned char * m;    // Declare pointer variable
    int R_MEAN = 33;      // Set the mean of red color of the target object
    int G_MEAN = 67;      // Set the mean of green color of the target object
    int B_MEAN = 161;     // Set the mean of blue color of the target object
    .......................
    m = cam_track_color(R_MEAN-30,R_MEAN+30,G_MEAN-30,G_MEAN+30,B_MEAN-
30,B_MEAN+30);
```

After that, Developers can read value from m[0], to m[7] to convert to array data type. Therefore :

m[0] contains the x position of middle mass (mx). This value indicates the trend that target object on the left or right side of the camera module.

- If the target object in front of camera modules, the value is 40.

- If the target object is left of camera modules, the value is lower than 40.

- If the target object is right of camera modules, the value is higher than 40.

m[1] contains the y position of middle mass (my). This value indicates the trend that target object on the top or bottom of the camera module.

- If the target object in front of camera modules, value is 71.

- If the target object is top of camera modules, value is lower than 71.

- If the target object is bottom of camera modules, value is higher than 71.

m[2] contains the x1 position; the left edge position in x axis

m[3] contains the y1 position; the left edge position in y axis

m[4] contains the x2 position; the right edge position in x axis

m[5] contains the y2 position; the right edge position in y axis

m[6] contains the number of pixel in track region

m[7] contains the confidence value. The maximum is 255. With this valus is more, it has more confident that the target color. This value may also be used as indicators of distance to the target object.

Note : Almost get only m[0] and m[7] to track the color of target object. such as the color of opposite goal in osccer robot competition.

7.2.4 MID_X and MID_Y constant

The camera moduel of ZX-CCD has 80 x 143 pixel resolution. The MID_X is middle positon of x axis value. The value is 40. The MID_Y is middle position of y axis value. It is equal 71.

Experiment 13 : Simple interfacing with ZX-CCD

This experiment demonstrates the ZX-CCD interfacing with Robo-128 to track the color composition of the target object and displays the mean value on the LCD screen. You will require a piece of 30 x 30cm. blue paper and yellow paper to represent the target object.

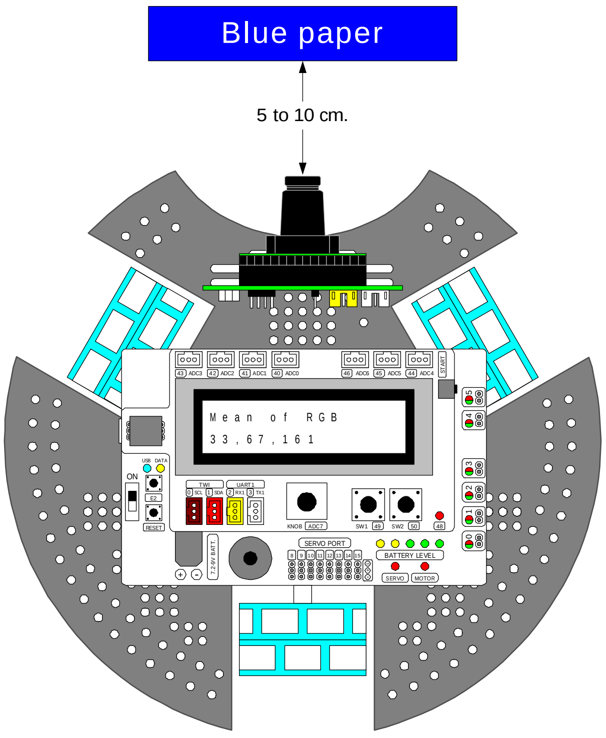
Why do we use the blue and yellow paper ? In the soccer robot competition, they define the color of goal of each side as BLUE and YELLOW. Thus, the blue and yellow color tracking is Robo-128's task in a soccer competition. The first step of this task is getting the mean value of blue and yellow object from ZX-CCD.

From this experiment, ZX-CCD is installed in front of the Robo-128. Connect Tx pin to RX port of Robo-128 and Rx pin to Tx port of Robo-128.

L13.1 Create a new sketch file and save it as cam_01. Type in the code following the Listing L13-1.

L13.2 Compile and upload the sketch file to Robo-128 robot.

L13.3 Set the blue paper in front of the robot and far about 5 to 10 cm.



```
#include <robot.h>
#include <cam.h>      // Include ZX-CCD library file
unsigned char * m;    // Declare pointer variable to access the ZX-CCD
void setup()
{
  cam_init();         // Initial the ZX-CCD operation
}
void loop()
{
  m = cam_get_mean();
          // Get the mean value of the target object from ZX-CCD
  lcd("Mean of RGB #n%d,%d,%d         ",m[0],m[1],m[2]);
          // Displays RGB mean value on the lower line of LCD screen
}
```

Listing L13-1 : cam_01.pde file; the C/C++ code of Wiring for getting RGB data from ZX-CCD of the Robo-128

L13.4 Run the code.

Robo-128's LCD shows message

Mean of RGB

RRR , GGG , BBB

Therefore

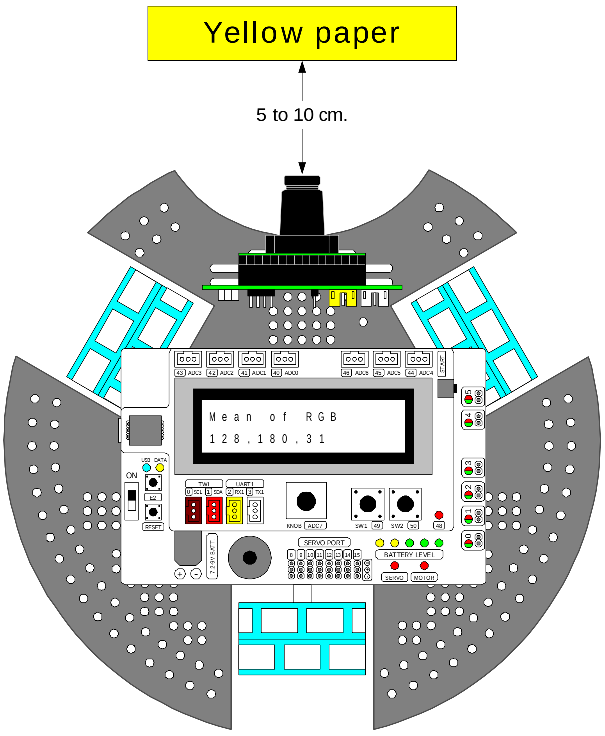
RRR is the mean value of red composition of the object

GGG is the mean value of green composition of the object

BBB is the mean value of blue composition of the object

With blue paper testing, BBB value is highest of all values.

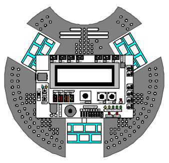
L13.5 Change the blue paper to yellow paper. Record the RRR, GGG and BBB values from the ZX-CCD.



The result value of RRR and GGG is higher than BBB value. Because yellow color has red and green color composition.

These are some examples of tracking the blue and yellow object.

Object	*Red composition*	*Green composition*	*Blue composition*
Blue paper	*33*	*67*	*161*
Yellow paper	*128*	*180*	*31*



Experiment 14 : Blue and Yellow color tacking

Experiment 14.1 - Blue object tracking

This experiment presents an example of C/C++ programming to control the Robo-128 to detect the blue object and displays the mean value to LCD screen. The interesting value are middle mass value and confidence value.The condition of blue color tracking are R_MEAN ±30, G_MEAN±30 and B_MEAN±30. The mean value of each composition are get from previous experiment as follows :

Object	*Red composition*	*Green composition*	*Blue composition*
Blue paper	*33*	*67*	*161*
Yellow paper	*128*	*180*	*31*

The hardware connection still same the experiment 13.

L14.1.1 Create a new sketch file and save as cam_02. Type in the code following the Listing L14-1.

```
#include <robot.h>
#include <cam.h>          // Include ZX-CCD library file
unsigned char * m;        // Declare pointer variable to access the ZX-CCD
unsigned char conf=0;     // Declare the confidence vairable for target
                          // color (it is blue in this code)
unsigned char mx=0;       // Declare the middle mass variable
int R_MEAN = 33;          // Set the mean value of red color
int G_MEAN = 67;          // Set the mean value of green color
int B_MEAN = 161;         // Set the mean value of blue color
void setup()
{
  cam_init();             // Initial ZX-CCD
}
void loop()
// Detect the target object by setting condition as
// R_MEAN+/-30, G_MEAN+/-30 and B_MEAN+/-30
{
  m = cam_track_color(R_MEAN-30,R_MEAN+30,G_MEAN-30,G_MEAN+30,B_MEAN-30,B_MEAN+30);
  conf = m[7];            // Store the confidence value
  mx = m[0];              // Store the middle mass value (mx)
  lcd("Conf: %d    #nmx: %d    ",conf,mx);
                          // Display the result at LCD module
}
```

Listing L14-1 : cam_02.pde file; the C/C++ code of Wiring for deteting the blue object of the Robo-128

L14.1.2 Compile and upload the sketch file to Robo-128 robot.

L14.1.3 Set the blue paper in front of the robot and far about 5 to 10 cm.

L14.1.4 Run the code

Robo-128's LCD shows message

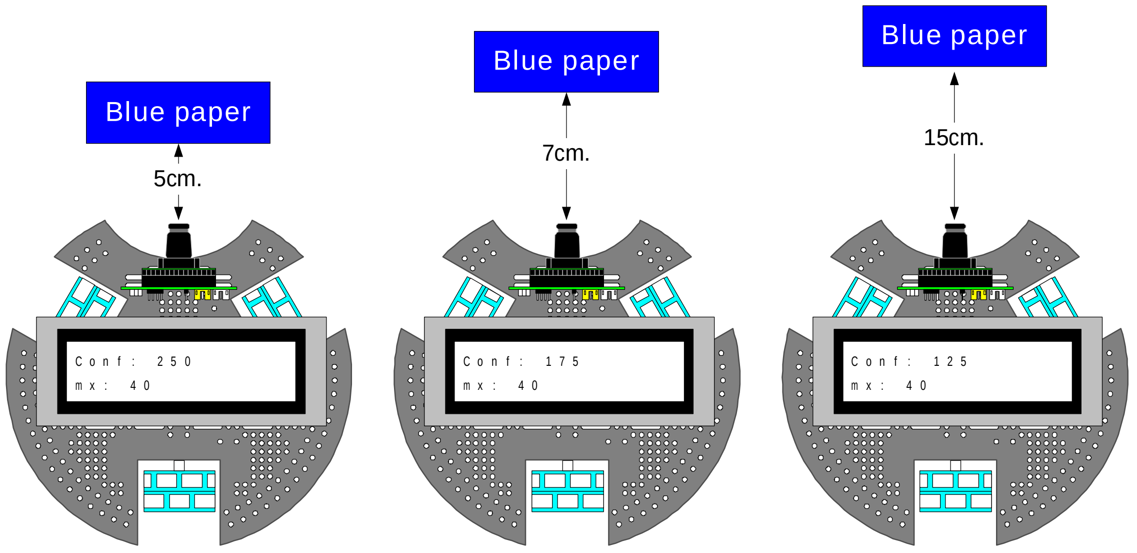
```
Conf :   aaa
mx :   bbb
```

Therefore

aaa is the confidence value of the target color

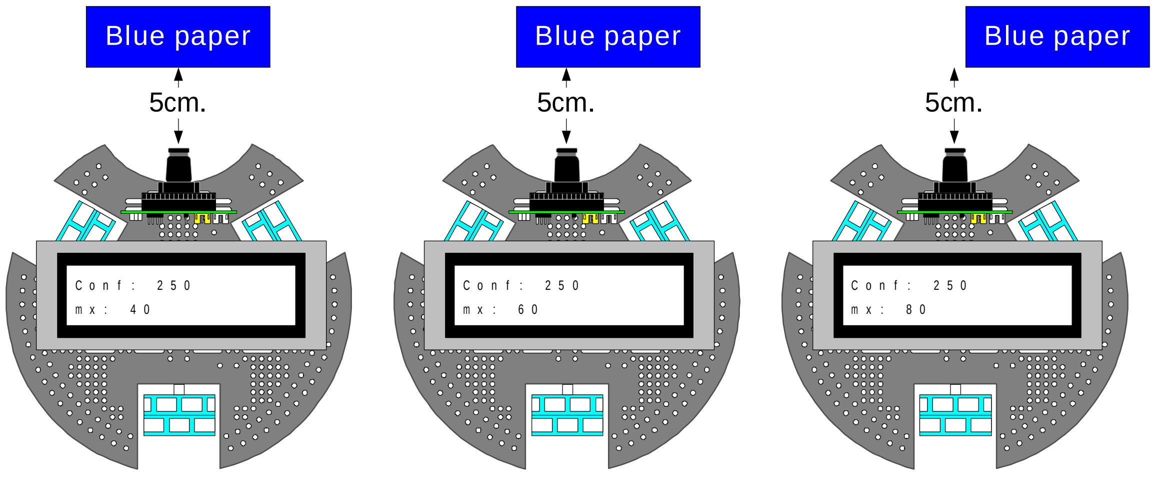
bbb is the middle mass value of x axis. This value represents the position of object in x axis.

L14.1.5 Move the blue paper with 5, 7 and 10cm. from the robot. Observe the robot's operation



From testing, the confidence value is high when the target color object near the sensor and decreade the value when far away.

L14.1.6 Shift the blue paper from left to right direction slowly. Observe the mx value.



The middle mass value of x axis (mx) is increasing until it reaches 40. The object position is center-front of the robot. The mx value still increaase following the object shift right out of sensor range. The maximum value is 80

L14.1.7 Change the blue paper to yellow and test again.

The result is both value (conf and mx) near 0. Because the yellow color is more different from blue color.

Experiment 14.2 - Yellow object tracking

This experiment is similar the previous experiment. Change the color target from blue to yellow.

L14.2.1 Create a new sketch file and save as cam_03. Type in the code following the Listing L14-3

L14.2.2 Compile and upload the sketch file to Robo-128 robot.

L14.2.3 Run the code

Robo-128's LCD shows message

```
Conf:  aaa
mx:   bbb
```

Therefore

aaa is the confidence value of the target color

bbb is the middle mass value of x axis. This value represents the position of object in x axis.

```
#include <robot.h>
#include <cam.h>          // Include ZX-CCD library file
unsigned char * m;        // Declare pointer variable to access the ZX-CCD
unsigned char conf=0;     // Declare the confidence vairable for target
                          // color (it is yellow in this code)
unsigned char mx=0;       // Declare the middle mass variable
int R_MEAN = 128;         // Set the mean value of red component for yellow
int G_MEAN = 180;         // Set the mean value of green component for yellow
int B_MEAN = 31;          // Set the mean value of blue componnet for yellow
void setup()
{
  cam_init();             // Initial ZX-CCD
}
void loop()
{
  // Detect the target object by setting condition as
  //  R_MEAN+/-30, G_MEAN+/-30 BÁ_MEAN+/-30
  m = cam_track_color(R_MEAN-30,R_MEAN+30,G_MEAN-30,G_MEAN+30,B_MEAN-30,B_MEAN+30);
  conf = m[7];            // Store the confidence value
  mx = m[0];              // Store the middle mass value (mx)
  lcd(“Conf: %d    #nmx: %d    “,conf,mx);
                          // Display the result at LCD module
}
```

Listing L14-2 : cam_03.pde file; the C/C++ code of Wiring for deteting the yellow object of the Robo-128

L14.2.4 Move the yellow sheet with 5, 7 and 10cm. from the robot. Observe the robot’s operation.

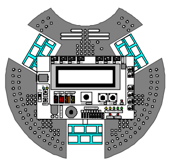
From the testing, the confidence value is high when the target color object near the sensor and decreae the value when far away.

L14.2.5 Shift the yellow sheet from left to right direction slowly. Observe the mx value.

The middle mass value of x axis (mx) is increasing until it is 40 the object position is center-front of the robot. the mx value still increaase following the object shift right out of sensor range. The maiimum value is 80

L14.2.6 Change the yellow sheet to blue and test again.

The result is both value (conf and mx) near 0. Because the blue color is more different from yellow color.



Experiment 15 : Blue navigator robot

This experiment demonstrares the example of C/C++ programming to control the Robo-128's monement by tracking the blue object. There is 3 conditions as follows :

1. The robot must attempt to move towards the blue paper on the front of the ZX-CCD.

2. If the blue paper is on the left or right of the ZX-CCD camera module, the robot must turn around to find a blue sheet.

3. If cannot found the blue sheet, robot must stop.

L15.1 Create a new sketch file and save as robot_cam_detect_blue. Type in the code following the Listing L15-1.

L15.2 Compile and upload the sketch file to Robo-128 robot.

L15.3 Place in front of the robot to the direction under test.

L15.4 Run the code

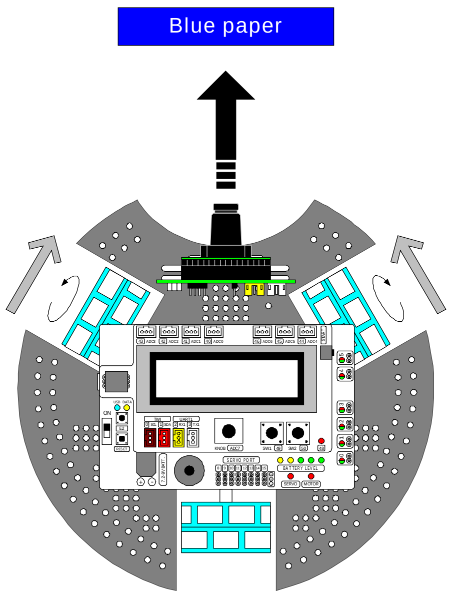
Robo-128's LCD shows message

```
SW1 Press
```

L15.5 Set the blue sheet in front of the robot and far about 5 to 10 cm.

L15.6 Press the SW1 and observe the robot's movement.

Robo-128 moves forward to the blue sheet.

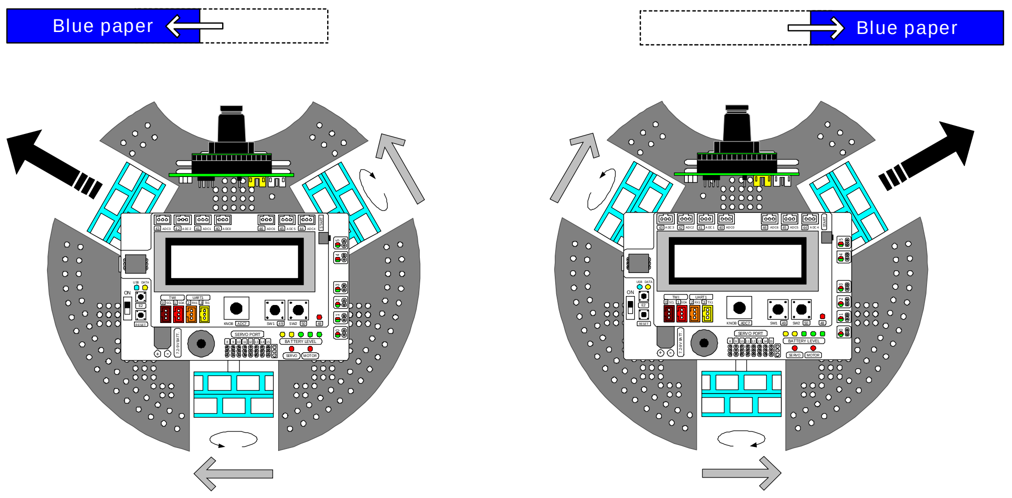


```
#include <robot.h>
#include <cam.h>        // Include ZX-CCD library file
unsigned char * m;      // Declare pointer variable to access the ZX-CCD
unsigned char conf=0;   // Declare the confidence vairable for target color ( blue)
unsigned char mx=0;     // Declare the middle mass variable
int R_MEAN = 33;        // Set the mean value of red component for blue
int G_MEAN = 67;        // Set the mean value of green component for blue
int B_MEAN = 161;       // Set the mean value of blue component for blue
void spin_left(int p)   // Spin left function
{
   motor(0,p);          // Drives the front-left motor backward by p power
   motor(1,p);          // Drives the back wheel motor to right direction by p power
   motor(2,p);          // Drives the front-right motor forward by p power
}
void spin_right(int p) // Spin right function
{
   motor(0,-p);         // Drives the front-left motor forward by p power
   motor(1,-p);         // Drives the back wheel motor to left direction by p power
   motor(2,-p);         // Drives the front-right motor backward by p power
}
void forward(int L,int R)// Move forward function
{
   motor(0,-L);         // Drives the front-left motor forward by L power
   motor(1,0);          // Stop the back wheel motor
   motor(2,R);          // Drives the front-right motor forward by R power
}
void setup()
{
  cam_init();           // Initial ZX-CCD
 lcd("SW1 Press!");     // Display message for pressing SW1
   sw1_press();         // Wait until the SW1 is pressed
 }
void loop()
{  // Detect the target by setting condition as R_MEAN+/-30, G_MEAN+/-30 BÁ_MEAN+/-30
   m = cam_track_color(R_MEAN-30,R_MEAN+30,G_MEAN-30,G_MEAN+30,B_MEAN-30,B_MEAN+30);
   conf = m[7];             // Store the confidence value
   mx = m[0];               // Store the middle mass value (mx)
   if(conf>20)              // Detect blue ?
   {
     if(mx<MID_X-5)         // Blue object is on the left ?
     {
        spin_left(30);      // If yes, spin left the robot to face the object.
     }
     else if(mx>MID_X+5)    //  Blue object is on the right ?
     {
        spin_right(30);     // If yes, spin right the robot to face the object
     }
     else                   // The obejct in front of the robot
     {
        forward(80,80);     // Move forward to object
     }
   }
   else                     // Do not detect the object
   {
     motor_stop(ALL);       // Robot stop
   }
   sleep(2);                // Delay for ZX-CCD
}
```

Listing L15-1 : robot_cam_detect_blue.pde file; the C/C++ code of Wiring for Robo-128 to move follows the blue object

L15.7 Shift the blue sheet from left to right direction of ZX-CCD. Observe the robot's operation.

The Robo-128 will spin to follows the blue sheet.



L15.7 Move the blue sheet far from the Robo-128 or to back of the robot. Observe the robot's operation.

Robo-128 stops This is becasue the blue sheet is out of range.