NEXUS ROBOT

Looking to the future

Robot Kits User's Manual

Nexus Automation Limited
ADDR: 2/F,Chengxi Building. 819 S358 Road. Shatou, Changan, Dongguan, Guangdong, China

Robot Kits User's Manual

Tel: +86-769-85301107

Nexus Robot

A. Attention!! Please read this manual carefully before applying power on the device.

B. Attention!! Do not use this device for military or medical purpose as they are not designed to.

C. Attention!! Do not use over-voltage power supply ! ensure stable power supply. if there is high-voltage pulse, may cause the micro-control module permanent damage !

D. Attention!! This product is not waterproof feature, please keep or use it in a dry environment ! Don't piled the weight on top of it .

➢ Arduino

➢ Introduction

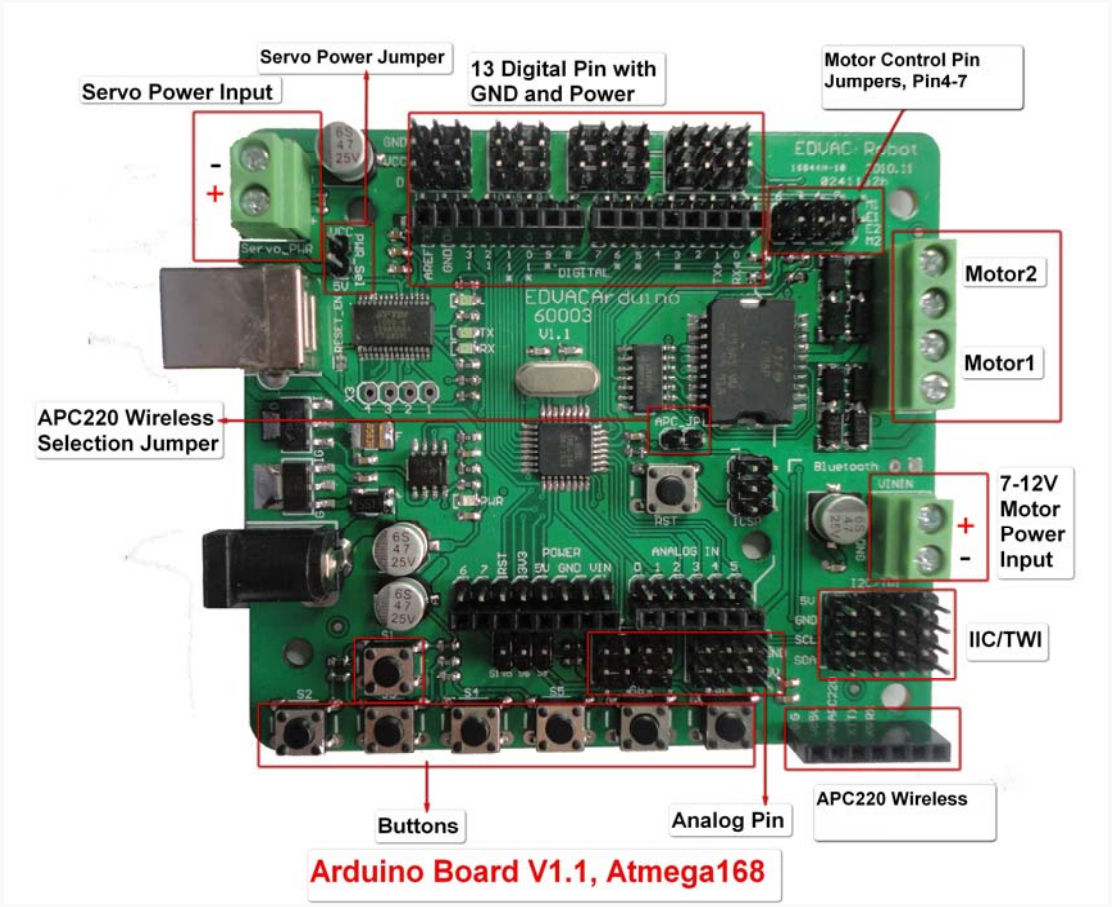
Arduino Controller is an All-in-One microcontroller especially designed for robotics application. Benefit from Arduino open source platform, it is supported by thousands of open source codes, and can be easily expanded with most Arduino Shields.Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. The microcontroller on the board is programmed using the Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). Arduino projects can be stand-alone or they can communicate with software running on a computer (e.g. Flash, Processing, MaxMSP).

➢ Hardware

➢ Specification

- Atmega 168/328
- 14 Channels Digital I/O
- 6 PWM Channels (Pin11,Pin10,Pin9,Pin6,Pin5,Pin3)
- 8 Channels 10-bit Analog I/O
- USB interface
- Auto sensing/switching power input
- ICSP header for direct program download
- Serial Interface TTL Level
- Support AREF
- Support Male and Female Pin Header
- Integrated sockets for APC220 RF Module
- Five IIC Interface Pin Sets
- Two way Motor Drive with 2A maximum current
- 7 key inputs
- DC Supply：USB Powered or External 7V~12V DC
- DC Output：5V /3.3V DC and External Power Output
- Dimension：90x80mm

➢ Arduino Board Pinout



The picture above shows all of the I/O lines and Connectors on the controller, which includes:

- One Regulated Motor Power Input Terminal (6v to12v)
- One Unregulated Servo Power Input Terminal (you supply regulated 4v to 7.2v)
- One Servo input power selection jumper
- One Serial Interface Module Header for APC220 Module
- Two DC Motor Terminals – Handles motor current draw up to 2A, each terminal
- One IIC/TWI Port – SDA, SCL, 5V, GND
- One Analog Port with 8 analog inputs – one input is tied internally to the supply voltage
- One General Purpose I/O Port with 13 I/O lines – 4,5,6,7 can be used to control motors
- One Reset Button
- Jumper bank to Enable/Disable Motor Control

➢ Arduino 328 Features

The Arduino Uno is a microcontroller board based on the ATmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

➢ Memory function

The ATmega328 has 32 KB (with 0.5 KB used for the bootloader). It also has 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library).

➢ Input and Output PinMode

Each of the 14 digital pins on the Duemilanove can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the FTDI USB-to-TTL Serial chip.

External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.

PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite() function.

SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.
LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

➢ Communication

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega8U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '8U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows, a .inf file is required. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

➢ Configuration

➢ Servo Power Select Jumper

As most servo draw more current than the USB power source can supply. A separate servo power terminal is provided to power the servo individually which can be Enable/Disable by the Servo Power Select Jumper.

When the Servo Power Select Jumper is applied, the servo is powered by internal 5V.

When the Servo Power Select Jumper is not applied, the servo is powered by external power source.

➢ Motor Control Pin Jumper

Applying the Motor Control Pin Jumpers will allocate Pin 4,5,6,7 for motor control.

Removing the jumpers will release the above Pins.

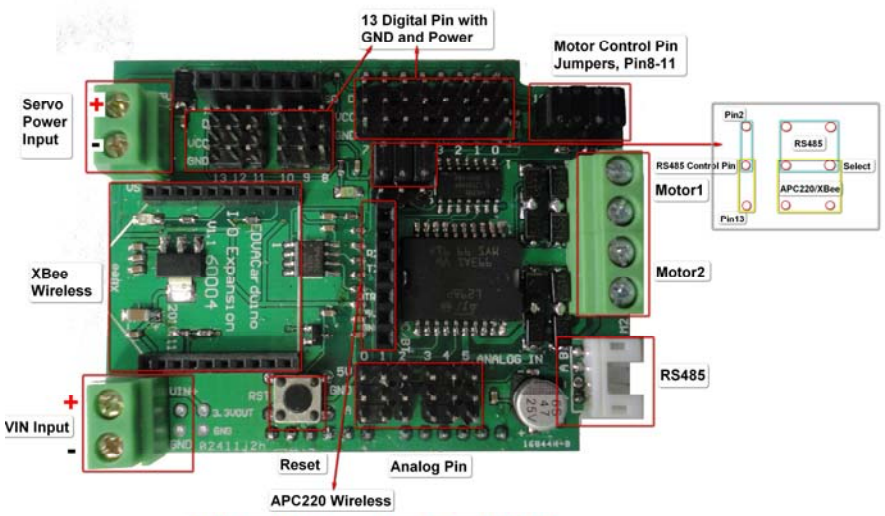
➢ Wireless Select Jumper

Applying the Wireless Select Jumper will allow the controller communicate via its wireless module such as APC220. If no wireless module is plugged, this jumper does not make any difference.

Removing the jumper will disable wireless module and allows the sketch to be uploaded.

➢ Arduino IO Expansion Board

To support RS485 interface or drive 4 motors, IO Expansion Board is available.



This Arduino compatible I/O EXpansion Sheild is intelligently designed to facilitate an easy connection between an Arduino board (e.g. Arduino Duemilanove) and other devices such as sensors and RS485 devcices. In essence, it expands an Arduino controller's Digital I/O and Analogue Input Pins with Power and GND. It is compatible with Arduino Mega and is a perfect companion of Arduino Dumilanove (Atmega168 and Atmega 328).

➢ Features

1. Supporting XBee (Xbee Pro)/Bluetooth Bee;
2. An unique RS485 output, supporting a RS485 device;
3. Separate PWM Pins, which are compatible with standard servo Connector;

4. Supporting Bluetooth module, APC220 module;

5. Auto Switch between external and onboard power supply;

6. Supporting SD card (read&write - our SD card module is needed);

7. Supporting IIC/I2C/TWI connection

See: http://www.nexusrobot.com/product.php?id_product=51

➢ Software

The open-source Arduino environment makes it easy to write code and upload it to the i/o board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing, avr-gcc, and other open source software.

➢ Before you start

Before you start there are somethings you will to attention.

➢ Applying Power

This is one of the most important steps in getting the controller up and communicating with your host controller. You must make sure that you apply power to the Power Terminal using the correct polarity. Reverse Polarity will damage the controller. *We are not responsible for such damage, nor do we warranty against such damage.* Make sure you take time to apply power correctly. Otherwise, it could get costly for you!

Power from USB: Simply plug USB cable, and the controller is able to work. Please notice that the USB can only supply 500 mA current. It should be able to meet the most requirements for LED lit application. However it is not enough to power DC motors or servo.

Power from Motor Power Input: Simply connect the ground wire from your supply to the screw terminal labeled “GND”, and then connect the positive wire from your supply to the screw terminal labeled “VIN".

NOTE: Maximum supply voltage cannot exceed 14V DC.

➢ Getting Started

This document explains how to connect your Arduino board to the computer and upload your first sketch.

1 | Get an Arduino board and USB cable

In this tutorial, we assume you're using an Arduino Uno, Arduino Duemilanove, Nano, or Diecimila. If you have another board, read the corresponding page in this getting started guide.

You also need a standard USB cable (A plug to B plug): the kind you would connect to a USB printer, for example. (For the Arduino Nano, you'll need an A to Mini-B cable instead.)

2 | Download the Arduino environment

Get the latest version from the download page.

When the download finishes, unzip the downloaded file. Make sure to preserve the folder structure.

Double-click the folder to open it. There should be a few files and sub-folders inside.

3 | Connect the board

The Arduino Uno, Mega, Duemilanove and Arduino Nano automatically draw power from either the USB connection to the computer or an external power supply. If you're using an Arduino Diecimila, you'll need to make sure that the board is configured to draw power from the USB connection. The power source is selected with a jumper, a small piece of plastic that fits onto two of the three pins between the USB and power jacks. Check that it's on the two pins closest to the USB port.

Connect the Arduino board to your computer using the USB cable. The green power LED (labelled **PWR**) should go on.

4 | Install the drivers

Installing drivers for the Arduino Uno with Windows7, Vista, or XP:

- Plug in your board and wait for Windows to begin it's driver installation process. After a few moments, the process will fail, despite its best efforts
- Click on the Start Menu, and open up the Control Panel.
- While in the Control Panel, navigate to System and Security. Next, click on System. Once the System window is up, open the Device Manager.
- Look under Ports (COM & LPT). You should see an open port named "Arduino UNO (COMxx)"
- Right click on the "Arduino UNO (COmxx)" port and choose the "Update Driver Software" option.
- Next, choose the "Browse my computer for Driver software" option.
- Finally, navigate to and select the Uno's driver file, named "ArduinoUNO.inf", located in the "Drivers" folder of the Arduino Software download (not the "FTDI USB Drivers" sub-directory).

Windows will finish up the driver installation from there.

See also: step-by-step screenshots for installing the Uno under Windows XP.

Installing drivers for the Arduino Duemilanove, Nano, or Diecimila with Windows7, Vista, or XP:

When you connect the board, Windows should initiate the driver installation process (if you haven't used the computer with an Arduino board before).

On Windows Vista, the driver should be automatically downloaded and installed. (Really, it works!)

On Windows XP, the Add New Hardware wizard will open:

When asked Can Windows connect to Windows Update to search for software? select No, not this time. Click next.

Select Install from a list or specified location (Advanced) and click next.

Make sure that Search for the best driver in these locations is checked; uncheck Search removable media; check Include this location in the search and browse to the drivers/FTDI USB Drivers directory of the Arduino distribution. (The latest version of the drivers can be found on the FTDI website.) Click next.

The wizard will search for the driver and then tell you that a "USB Serial Converter" was found. Click finish.

The new hardware wizard will appear again. Go through the same steps and select the same options and location to search. This time, a "USB Serial Port" will be found.

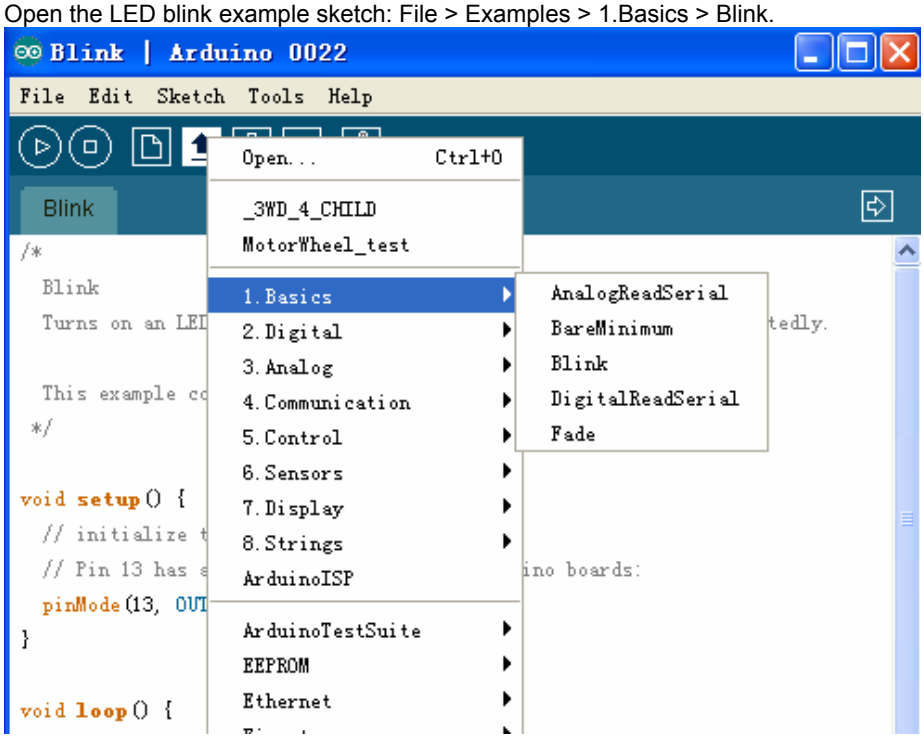
You can check that the drivers have been installed by opening the Windows Device Mananger (in the Hardware tab of System control panel). Look for a "USB Serial Port" in the Ports section; that's the Arduino board.

5 | Launch the Arduino application

Double-click the Arduino application.

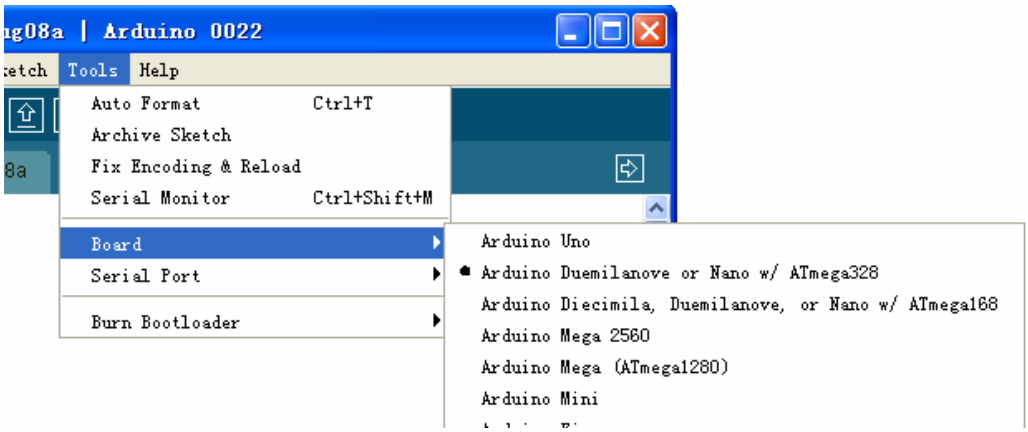
6 | Open the blink example

Open the LED blink example sketch: File > Examples > 1.Basics > Blink.



7 | Select your board

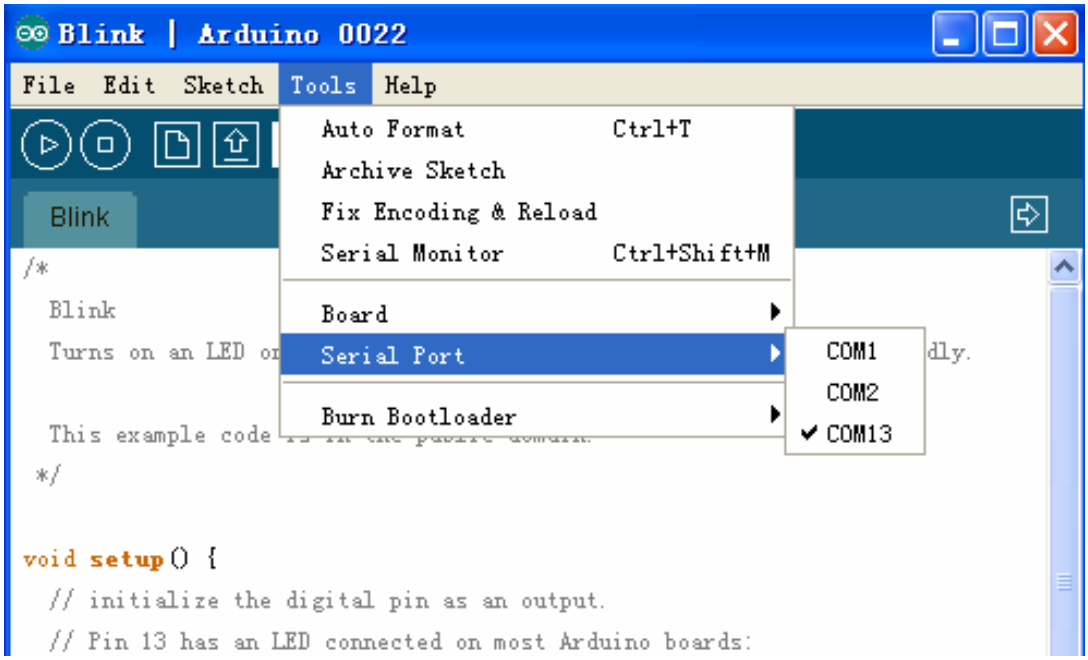
You'll need to select the entry in the **Tools > Board** menu that corresponds to your Arduino.



For Duemilanove Arduino boards with an ATmega328 (check the text on the chip on the board), select **Arduino Duemilanove or Nano w/ ATmega328**. Previously, Arduino boards came with an ATmega168; for those, select **Arduino Diecimila, Duemilanove, or Nano w/ ATmega168**. (Details of the board menu entries are available on the environment page.)

8 | Select your serial port

Select the serial device of the Arduino board from the Tools | Serial Port menu. This is likely to be **COM3** or higher (**COM1** and **COM2** are usually reserved for hardware serial ports). To find out, you can disconnect your Arduino board and re-open the menu; the entry that disappears should be the Arduino board. Reconnect the board and select that serial port.



9 | Upload the program

Now, simply click the "Upload" button in the environment. Wait a few seconds - you should see the RX and TX leds on the board flashing. If the upload is successful, the message "Done uploading." will appear

in the status bar. (*Note:* If you have an Arduino Mini, NG, or other board, you'll need to physically present the reset button on the board immediately before pressing the upload button.)

A few seconds after the upload finishes, you should see the pin 13 (L) LED on the board start to blink (in orange). If it does, congratulations! You've gotten Arduino up-and-running.

If you have problems, please see the troubleshooting suggestions.

You might also want to look at:

the examples for using various sensors and actuators

the reference for the Arduino language

The text of the Arduino getting started guide is licensed under a Creative Commons Attribution-ShareAlike 3.0 License. Code samples in the guide are released into the public domain.

See: http://arduino.cc/en/Guide/Windows

➢ Language Reference

Arduino programs can be divided in three main parts: *structure*, *values* (variables and constants), and functions.If you want to understand more, please **See**: http://www.arduino.cc/en/Reference/HomePage

➢ Programming structure

This section describes the two important structures in the basic Arduino: setup () and loop ().They are indispensable.

The setup() function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. The setup function will only run once, after each power up or reset of the Arduino board.

After creating a setup() function, which initializes and sets the initial values, the loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

Sample code

```
  int buttonPin = 3;

// setup initializes serial and the button pin
void setup()
{
   beginSerial(9600);
   pinMode(buttonPin, INPUT);
}

// loop checks the button pin each time,
// and will send serial if it is pressed
void loop()
```

```
{
  if (digitalRead(buttonPin) == HIGH)
    serialWrite('H');
  else
    serialWrite('L');

  delay(1000);
}
```

➢ Re-write Arduino bootloader

If you couldn't load the bootloader via the Arduino IDE with the parallel programmer from the Arduino website. Then you can used the following method to Re-write the bootloader on your chip.

➢ FT232RL BitBang Mode AVR-Writer

FT232RL is an USB-Serial bridge on an Arduino Dicimila/NG/Duemilanove PCB. It has the function to manipulate each signal pin directly. It's called BitBang Mode.
If we use "avrdude-serjtag" we can burn the bootloader by Diecimila itself.
This section describe the method on Windows-XP.

Attention!!!

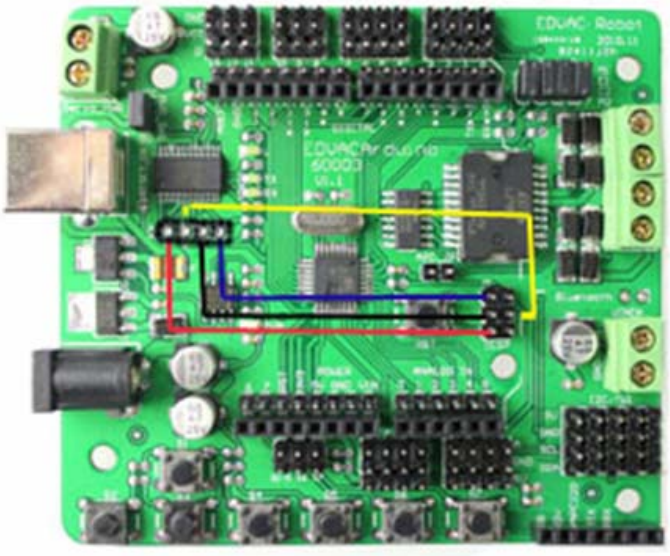
If you want use "avrdude-serjtag" on Linux or Mac OS, you must remake (patch, reconfigure and recompile) it.

There are useful projects. avrdude by FT245R/FT232R(Linux) and BitBang Mode AVR-Writer on Mac (Translated to English by Google.) These projects made a way to BitBang Mode AVR-Writer on Linux and Mac !!!

Of course, if you use Windows in Vmware on your Linux or Mac OS, you can run "avrdude-serjtag(windows version)".

➢ Modify the Diecimila

There are four pads written as X3 near FT232RL on a Diecimila PCB. (These pads are connected to the control pins of FT232RL.).Remove the solder of these pads and insert a pin-header. And soldering it.Connect the pins of X3 and the pins of ICSP by wires. Please see the photograph below. (click to enlarge)



➢ Downloading

To downloading the "avrdude-serjtag" FTDI BitBang AVR-Writer from the internet.

configure-file for avrdude-serjtag

avrdude.conf (**Update:** included chip-parameter of ATmega328P and 88P)

avrdude-GUI-1.0.5.zip mirror site

avrdude-GUI-1.0.5.zip original site

(http://yuki-lab.jp/hw/avrdude-GUI/index.html)avrdude-GUI-1.0.5.zip mirror site

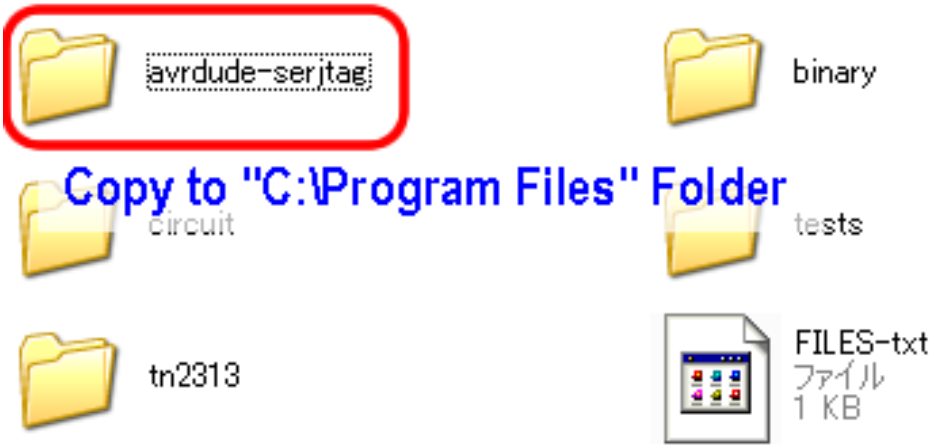
avrdude-GUI (yuki-lab.jp Version) require Microsoft .NET Framework 2.0. When .NET Framework 2.0 is not installed. Download it from here (Microsoft .NET Framework 2.0 download page) and install it.

➢ Installing

avrdude-serjtag

Exract serjtag-***.zip.

Copy "avrdude-serjtag" folder into the "C:\Program Files" folder.



Delete "src" folder in the "avrdude-serjtag" folder.

avrdude.conf

Copy(overwrite) "avrdude.conf" into the "C:\Program Files\avrdude-serjtag\binary" folder.

This modified "avrdude.conf" has setting-scripts of "FTDI BitBang AVR-Writer" for Diecimila below.

```
#arduino diecimila
Programmer
  id="diecimila";
  desc = "FT232R Synchronous BitBang";
  type = ft245r;
  miso = 3;   # CTS X3(1)
  sck = 5;   # DSR X3(2)
  mosi = 6;   # DCD X3(3)
  reset = 7;   # RI X3(4)
;
```

avrdude-GUI (yuki-lab.jp Version)

Extract avrdude-GUI-1.0.5.zip.

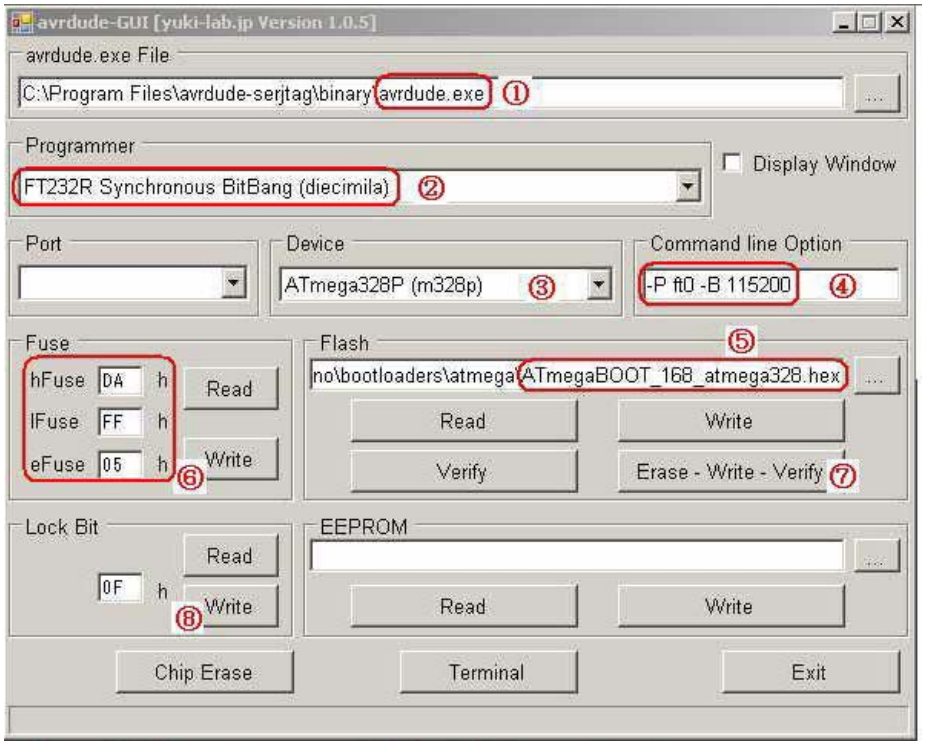
Copy "avrdude-GUI-1.0.5" folder into "C:\Program Files" folder.

avrdude-GUI (yuki-lab.jp Version) require Microsoft .NET Framework 2.0.

When .NET Framework 2.0 is not installed. Download it from here and install it.

➢ Setting

Open the "C:\Program Files\avrdude-GUI-1.0.5" folder. And double click the "avrdude-GUI.exe" to run it."avrdude-GUI" settings is as below from ① to ⑧.



If your chip isn't the same ,Some of the blank's setting is different .change it base on the following chips :

Microsoft .Net Framework 2.0 or higher required

Burning the Arduino bootloader without external AVR-Writer:

http://www.geocities.jp/arduino_diecimila/bootloader/index_en.html

```
Arduino Duemilanove or Nano w/ ATmega328
  hFuses=0xDA lFuses=0xFF eFuses=0x05 lockBit=0x0F
  flash=arduino-<xxxx>\hardware\arduino\bootloader\atmega\ATmegaBOOT_168_atmega328.hex
Arduino Diecimila, Duemilanove, or Nano w/ ATmega168
  hFuses=0xdd lFuses=0xff eFuses=0x00 lockBit=0x0F
  flash=arduino-<xxxx>\hardware\arduino\bootloader\atmega\ATmegaBOOT_168_diecimila.hex
Arduino Mega (ATmega1280)
  hFuses=0xDA lFuses=0xFF eFuses=0xF5 lockBit=0x0F
  flash=arduino-<xxxx>\hardware\arduino\bootloader\atmega\ATmegaBOOT_168_atmega1280.hex
Arduino Mega 2560
  hFuses=0xD8 lFuses=0xFF eFuses=0xFD lockBit=0x0F
  flash=arduino-<xxxx>\hardware\arduino\bootloader\stk500v2\stk500boot_v2_mega2560.hex
```

➢ Testing and confirming

- Disconnect a USB cable from Diecimila.
- Remove the wires of ICSP and X3.
- Connect a USB cable to Diecimila.
- Push the reset button of Diecimila.
- Start Arduino-IDE.
- Upload sample sketch "Blink".
- And it will be run

See: http://www.roboticfan.com/article/html/797.shtml

or http://www.geocities.jp/arduino_diecimila/bootloader/index_en.html

➢ Adjust PWM frequencies

The ATmega328P has three timers known as Timer 0, Timer 1, and Timer 2. Each timer has two output compare registers that control the PWM width for the timer's two outputs: when the timer reaches the compare register value, the corresponding output is toggled. The two outputs for each timer will normally have the same frequency, but can have different duty cycles (depending on the respective output compare register).

By macegr in this forum post

http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1235060559/12

Pins 5 and 6: controlled by timer 0

Setting	Divisor	Frequency
0x01	1	62500

0x02	8	7812.5
0x03	64	976.5625
0x04	256	244.140625
0x05	1024	61.03515625

TCCR0B = TCCR0B & 0b11111000 | <setting>;

Pins 9 and 10: controlled by timer 1

Setting	Divisor	Frequency
0x01	1	31250
0x02	8	3906.25
0x03	64	488.28125
0x04	256	122.0703125
0x05	1024	30.517578125

TCCR1B = TCCR1B & 0b11111000 | <setting>;

Pins 11 and 3: controlled by timer 2

Setting	Divisor	Frequency
0x01	1	31250
0x02	8	3906.25
0x03	32	976.5625
0x04	64	488.28125
0x05	128	244.140625
0x06	256	122.0703125
0x07	1024	30.517578125

TCCR2B = TCCR2B & 0b11111000 | <setting>;

All frequencies are in Hz and assume a 16000000 Hz system clock.

From koyaanisqatsi in this forum post

http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1235060559/12

If you change TCCR0B, it affects millis() and delay(). They will count time faster or slower than normal if you change the TCCR0B settings. Below is the adjustment factor to maintain consistent behavior of these functions:

Default: delay(1000) or 1000 millis() ~ 1 second

0x01: delay(64000) or 64000 millis() ~ 1 second

0x02: delay(8000) or 8000 millis() ~ 1 second

0x03: is the default

0x04: delay(250) or 250 millis() ~ 1 second

0x05: delay(62) or 62 millis() ~ 1 second

(Or 63 if you need to round up. The number is actually 62.5)

Also, the default settings for the other timers are:

TCCR1B: 0x03

TCCR2B: 0x04

There may be other side effects from changing TCCR0B. For example my project would not properly run with TCCR0B set to 0x02 or 0x01. But it worked fine at 0x03 and higher.

➢ Simple Examples in Arduino 328

These examples are designed to demonstrate how to use the modules with the Arduino. The Arduino's hardware serial port is not used to connect to our modules, which keeps it available to the USB port. That allows downloading new programs without having to continually disconnect/reconnect things. Most of these examples use the LCD03 display module to show the results, but it is also possible to display the results on the PC, as demonstrated in the CMPS03 example. All the modules which use the I2C bus have 1k8 pull-up resistors to 5v. You only need one set of resistors, located near the Arduino, regardless of however many I2C devices you have connected to it.

➢ LED control

Most Arduino boards already have an LED attached to pin 13 on the board itself. If you run this example with no hardware attached, you could see LED blinks.This example shows the simplest thing you can do with an Arduino to see physical output: it blinks an LED.

Sample code

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.
   This example code is in the public domain.
 */
void setup() {
  // initialize the digital pin as an output.Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}
void loop() {
  digitalWrite(13, HIGH);     // set the LED on
  delay(1000);                // wait for a second
  digitalWrite(13, LOW);      // set the LED off
  delay(1000);                // wait for a second
}
```

Led control

➢ Button module

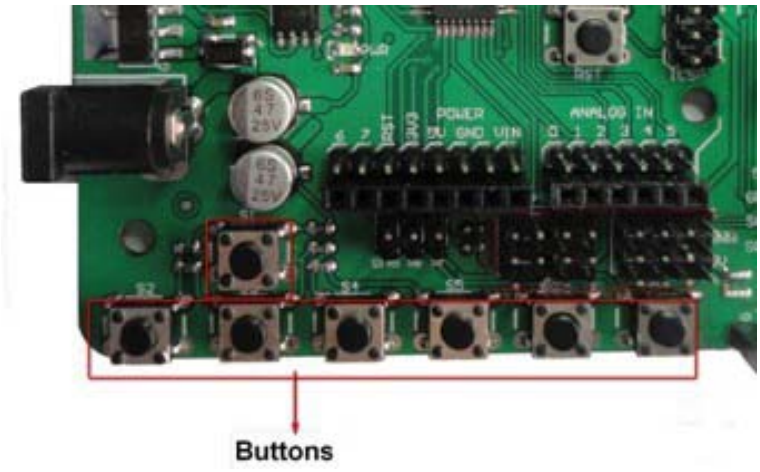
The controller has 7 build-in buttons S1-S7. S1-S5 use analog input, S6, S7 use digital input.

To enable S6 and S7, please apply the jumpers indicated in the red circle. S6 uses Digital Pin2, S7 uses Digital Pin3. Once these enable jumpers have been applied, Pin 2 and 3 will be occupied.

Sample code

```
int ledPin = 13;
int key_s6 = 2;
int val=0;
void setup()
{
  pinMode(ledPin, OUTPUT);       // Set Pin13 to output mode
  pinMode(key_s6, INPUT);        // Set Pin12 to output mode
}
void loop()
{
   if(digitalRead(key_s6)==0)     //
     {
        while(!digitalRead(key_s6));
        val++;
      }
   if(val==1) {
digitalWrite(ledPin, HIGH);    //
     }
   if(val==2)
     {
        val=0;
```

```
      digitalWrite(ledPin, LOW);     // Set a low value to the ledPin
   }
}
```



Arduino button

➢ Interrrupt control

Description

Specifies a function to call when an external interrupt occurs. Replaces any previous function that was attached to the interrupt. Most Arduino boards have two external interrupts: numbers 0 (on digital pin 2) and 1 (on digital pin 3). The Arduino Mega has an additional four: numbers 2 (pin 21), 3 (pin 20), 4 (pin 19), and 5 (pin 18).

Sample code

```
int pin = 13;
volatile int state = LOW;
void setup()
{
   pinMode(pin, OUTPUT);
   attachInterrupt(0, blink, CHANGE);
}
void loop()
{
   digitalWrite(pin, state);
}
void blink()
{
   state = !state;
}
```

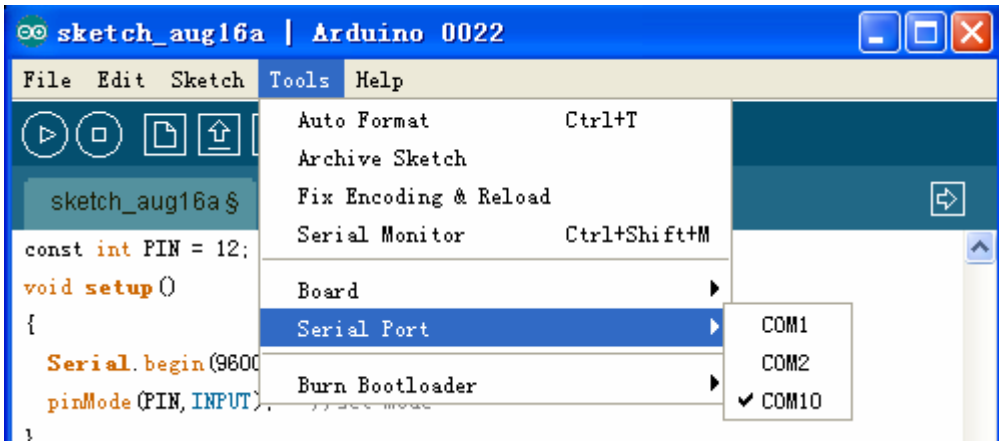
➢ Digital Read Serial

Fall Detector Example

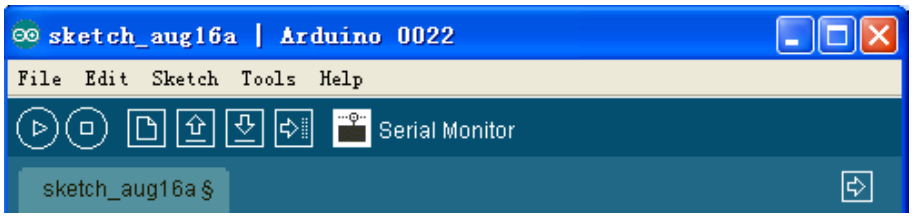
This example shows you how to monitor the state of a Fall detector by establishing serial communication between your Arduino and your computer over USB.

Sample code

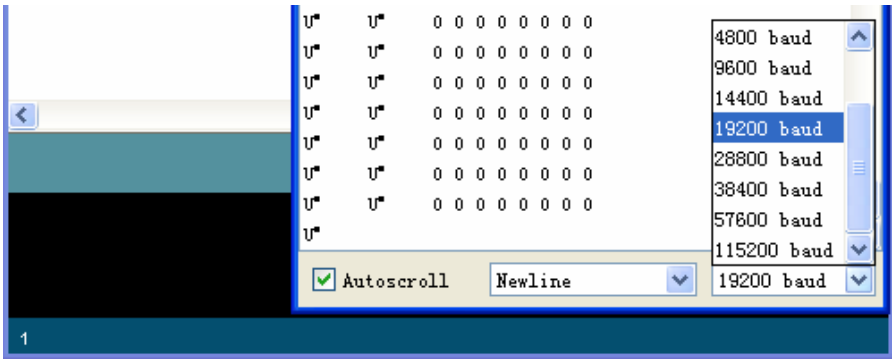
```
const int PIN = 12;    //set pin 12 as the signal pin
void setup()
{
   Serial.begin(9600);
   pinMode(PIN,INPUT);     //set mode
}
void loop()
{
   bool    val = 0;
   val = digitalRead(PIN);     //read pin 12
   Serial.println(val);           //display the value
   delay(500);
}
```



Chooce the serial port



Open the serial Monitor



Select the display Baud Rate

➢ Analog Read Serial

Sharp 2D12 Example

This example shows you how to read analog input, which from the physical world using a Sharp 2D12. A Sharp 2D12 is a simple mechanical device that provides a varying amount of resistance when its shaft is turned. By passing voltage through a Sharp 2D12 and into an analog input on your Arduino, it is possible to measure the amount of resistance produced by a Sharp 2D12 (or pot for short) as an analog value. In this example you will monitor the state of your Sharp 2D12 after establishing serial communication between your Arduino and your computer.

Sample code

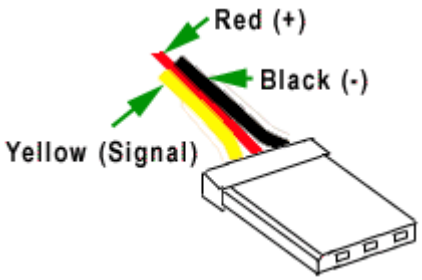
```
const int GP2Y0A21 = 0;     //set analog pin 0 as the signal pin
void setup()
{
   Serial.begin(9600);
}
void loop()
{
   int val = 0;
   val = ((10485/(analogRead(ISRpin[0])+5))-4);    //read the data from signal pin
   Serial.println(val,DEC);     // display
   delay(500);
}
```

➢ Servo Motor Theory

Servo motors have three wires: power, ground, and signal. The power wire is typically red, and should be connected to the 5V pin on the Arduino board. The ground wire is typically black or brown and should be connected to a ground pin on the Arduino board. The signal pin is typically yellow, orange or white and should be connected to a digital pin on the Arduino board. Note servos draw considerable power, so if

you need to drive more than one or two, you'll probably need to power them from a separate supply (i.e. not the +5V pin on your Arduino). Be sure to connect the grounds of the Arduino and external power supply together. As mentioned earlier, most servos expect a pulse width between 1-2 ms, however, a range of 0.5 ms to 2.5 ms (500-2500µs) may be required, depending on your servo. Experiment as necessary.

Hi-Tec Servo Motors have three wires coming out of them.

·Red - Power (4.8v-6v) ·Black (Ground) ·Yellow (Signal)	

The power & ground wires are hooked directly up to whatever battery or power supply you are using to power the servos. The Signal wire will be hooked up to the microcontroller used to control the servo, in our case the PIC. A noticeable first impression, the servo only requires 1 pin from the pic.

The PWM Signal

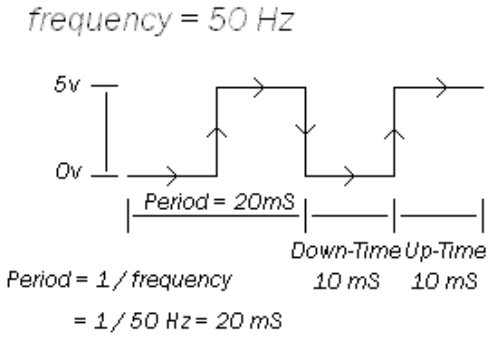
The signal that we need to create inorder to control the servos is called a Pulse With Modulation signal or PWM for short. The general requirements are:

Frequency: 50Hz

Up-time: 0.9mS->2.1mS

Down-time: 19.1mS-17.9mS

At first glance these definitions & numbers might make little or no sense. So lets look at a simple PWM wave at 50Hz.



So a PWM wave is just a signal that changes between 0 volts & 5 volts (digital logic 0 and 1). We see that the wave is symmetrical; uptime is 10mS & downtime is 10mS which when added together give us the period (10mS + 10mS)

See: http://www.pyroelectro.com/tutorials/servo_motor/index.html

➢ Motor Control

Hardware Setting

Connect four motor wires to Motor Terminal. And apply power through motor power terminal.

The PWM DC motor control is implemented by manipulating two digital IO pins and two PWM pins. As illustrated in the diagram above, Pin 4,7 are motor direction control pins, Pin 5,6 are motor speed control pins.

Pin Allocation

PWM Control Mode

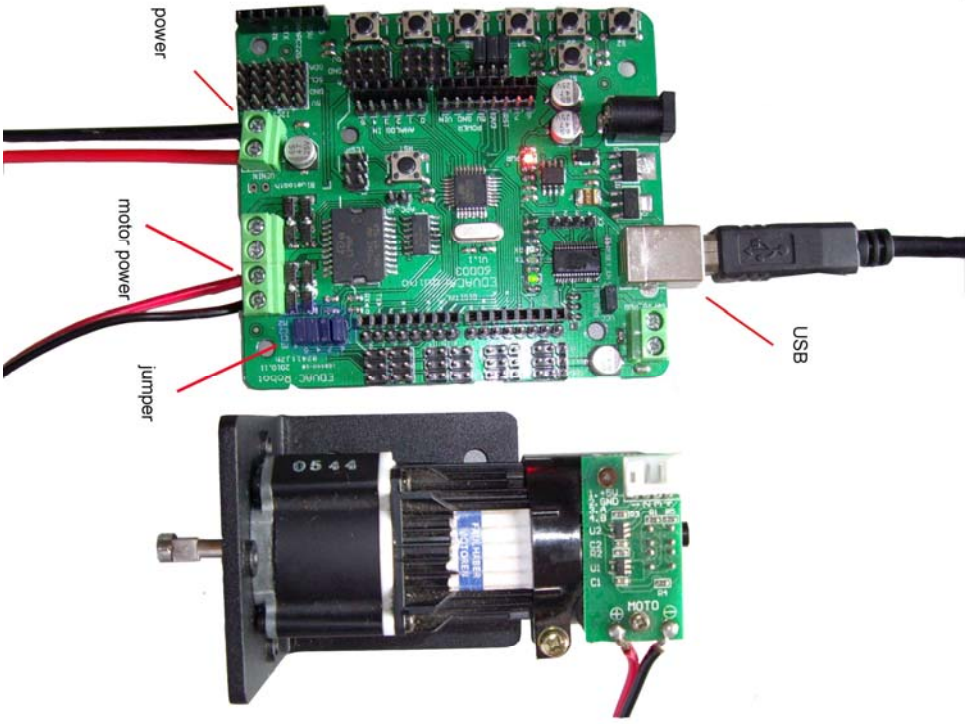
"PWM Mode"	
Pin	**Function**
Digital 4	Motor 1 Direction control
Digital 5	Motor 1 PWM control
Digital 6	Motor 2 PWM control
Digital 7	Motor 2 Direction control

"PLL Mode"	
Pin	**Function**
Digital 4	Motor 1 Enable control
Digital 5	Motor 1 Direction control
Digital 6	Motor 2 Direction control
Digital 7	Motor 2 Enable control

Sample code

```
  int E1= 6;         //the pin to control mator's speed
  int M1= 7;         //the pin to control direction
void setup()
{
  pinMode(M1,OUTPUT);    //M2 direction control
  pinMode(E1,OUTPUT);     //E2 PWM speed control
```

```
   analogWrite(E1,100);
  TCCR2B = TCCR2B & 0b11111000 | 0x01;
                                          //set the timer1 as the work intrrupt timer
                                          // to use the timer will defualt at the function of setup;
}
void loop() { }
```



The Motor sample to link wries

➢ Serial Port

This example shows you how to monitor the state of a switch by establishing serial communication between your Arduino and your computer over USB.

Sample code

```
/*
    DigitalReadSerial . Reads a digital input on pin 0, prints the result to the serial monitor
  Use the example of gp2y0A21.to see how the Serial work.
    */
const int GP2Y0A21 = 0;     //set analog pin 0 as the signal pin
int incomingByte = 0;        // for incoming serial data
void setup()
{
  Serial.begin(9600); // opens serial port, sets data rate to 9600 bps
}
```

```
void loop()
{
   incomingByte = ((10485/(analogRead(ISRpin[0])+5))-4);    //read the data from signal pin
   Serial.println(incomingByte,DEC);      // display
   delay(500);
}
```

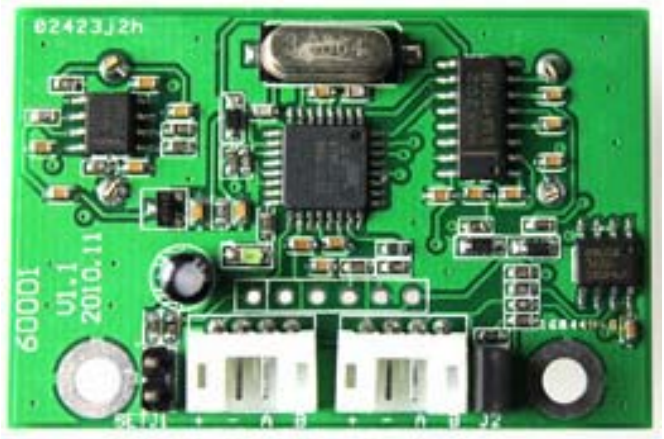
➢ External device modules

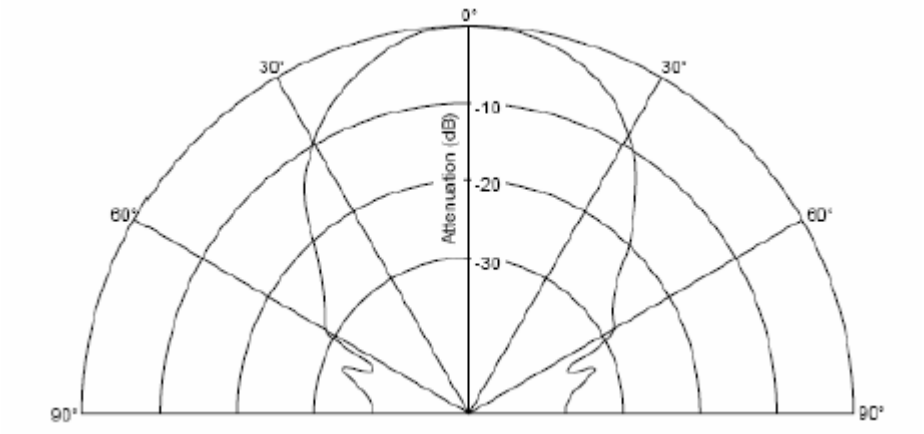
➢ Dual Ultrasonic Sensor (DUS)

➢ Introduction

DUS is based on RS485 interface. It allows a number of sensors working together. Up to 32 units may be connected together in a RS485 network.

The ultrasonic sensor allows you to determine the exact distance of an obstacle in the sonar field of view. The cleverness of your robot will depend on a sensitive sensor similar to the one bats use to know their position and track prey. With advanced programming, you can design compensators in order to perfectly control your motors according to the obstacle's distance.





➢ Specification

- Power: +5V
- Current: <20mA
- Working temperature: -10℃～+70℃
- Detecting range: 4cm-300cm
- Resolution: 1cm
- Frequency: 40KHz
- Interface: RS485
- Units: Range reported in cm
- Temperature sensor: 12 bits reading from serial port
- Size: 34mm × 51 mm
- Weight: 30g

➢ Dimension and Pin definition

RS485 Interface：Two connectors, +： +5V DC Power +5V, **-：** GND Ground , **A：** A RS485 A(+) , **B：** B RS485 B(-), **ISP Pin：**For factory firmware uploading

Communication LED：As the device is powered up, this LED will flash four times which indicates that the sensor is working properly. This LED will also flash when it is communicating with other devices. **Jumper A：**Not in use **Jumper B：**When the sensor is working under a network, only the Jumper B for the first Device and the last Device need to be bridged.

➢ RS485 Bus

➢ Introduction to RS485

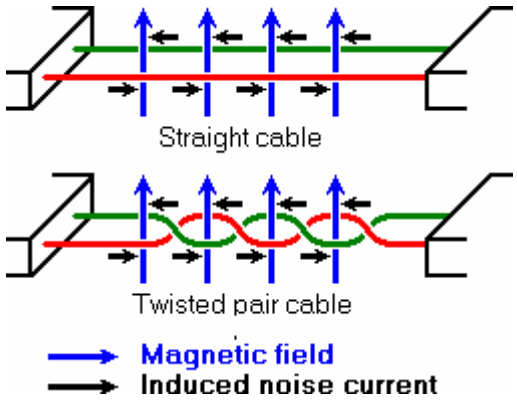
RS485 are serial communication methods for computers and devices.
RS485 bus advances in simpler cabling,longer transmitting distance and higher dependability.
RS485 is the most versatile communication standard in the standard series defined by the EIA, as it performs well on all four points.

➢ Differential signals with RS485

Longer distances and higher bit rates

As we know noise is easily picked up and limits both the maximum distance and communication speed. With RS485 on the contrary there is no such thing as a common zero as a signal reference. Several volts difference in the ground level of the RS485 transmitter and receiver does not cause any problems.

Noise in straight and twisted pair cables

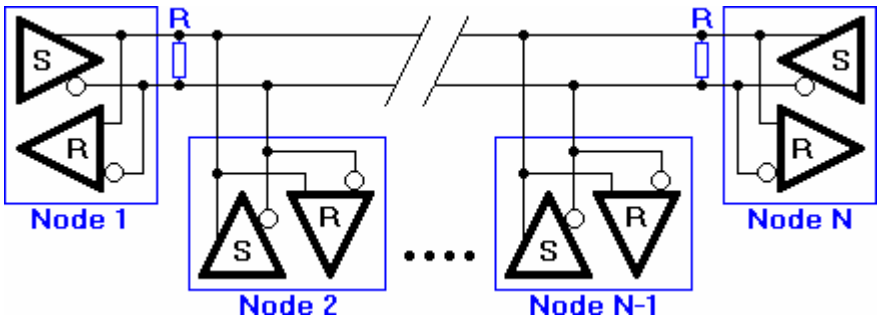


In the picture above, noise is generated by magnetic fields from the environment. The picture shows the magnetic field lines and the noise current in the RS485 data lines that is the result of that magnetic field. In the straight cable, all noise current is flowing in the same direction, practically generating a looping current just like in an ordinary transformer. When the cable is twisted, we see that in some parts of the signal lines the direction of the noise current is the oposite from the current in other parts of the cable. Because of this, the resulting noise current is many factors lower than with an ordinary straight cable. Shielding—which is a common method to prevent noise in RS232 lines—tries to keep hostile magnetic fields away from the signal lines. Twisted pairs in RS485 communication however adds immunity which is a much better way to fight noise. The magnetic fields are allowed to pass, but do no harm. If high noise immunity is needed, often a combination of twisting and shielding is used as for example in STP, shielded twisted pair and FTP, foiled twisted pair networking cables. Differential signals and twisting allows RS485 to communicate over much longer. With RS485 communication distances of 1200 m are possible.
Differential signal lines also allow higher bit rates than possible with non-differential connections. Currently RS485 drivers are produced that can achieve a bit rate of 35 mbps.

➢ Network topology with RS485

RS485 is the only of the interfaces capable of internetworking multiple transmitters and receivers in the same network. When using the default RS485 receivers with an input resistance of 12 kΩ it is possible to connect 32 devices to the network. Currently available high-resistance RS485 inputs allow this number to be expanded to 256. RS485 repeaters are also available which make it possible to increase the number of nodes to several thousands, spanning multiple kilometers. And that with an interface which does not require intelligent network hardware

RS485 network topology



In the picture above, the general network topology of RS485 is shown. N nodes are connected in a multipoint RS485 network. For higher speeds and longer lines, the termination resistances are necessary on both ends of the line to eliminate reflections. Use 100 Ω resistors on both ends. The RS485 network must be designed as one line with multiple drops, not as a star. Although total cable length maybe shorter in a star configuration, adequate termination is not possible anymore and signal quality may degrade significantly.

➢ RS485 functionality

Default, all the senders on the RS485 bus are in tri-state with high impedance. In most higher level protocols, one of the nodes is defined as a master which sends queries or commands over the RS485 bus. All other nodes receive these data. Depending of the information in the sent data, zero or more nodes on the line respond to the master. In this situation, bandwidth can be used for almost 100%. There are other implementations of RS485 networks where every node can start a data session on its own. This is comparable with the way ethernet networks function. Because there is a chance of data collosion with this implementation, theory tells us that in this case only 37% of the bandwidth will be effectively used. With such an implementation of a RS485 network it is necessary that there is error detection implemented in the higher level protocol to detect the data corruption and resend the information at a later time.

There is no need for the senders to explicity turn the RS485 driver on or off. RS485 drivers automatically return to their high impedance tri-state within a few microseconds after the data has been sent. Therefore it is not needed to have delays between the data packets on the RS485 bus.

See: http://www.lammertbies.nl/comm/info/RS-485.html#intr

➢ Communication Protocols

The device is fixed at 19200 bps Baud Rate,8/N/1.

Set Device Address

Command:

Header		Address	Length	Cmd	Set Address	SUM
55	Aa	AB	1	55	ADD	SUM

Return Value：

Header		Address	Length	Cmd	Flag	SUM
55	Aa	ADD	1	55	S	SUM

PS: All connected SONAR will be changed to the given address. The new address must be between 0x11 and 0x30. If the address is set successfully, the flag will be set to 0x01 in the return data. If unsuccessful, there is no return data. (The default address for the sensor is 0x11)

Example:

Command:

0x55 0xaa 0x11 0x01 0x55 0x12 0x79 (Set Address to 0x12)

Return:

0x55 0xaa 0x12 0x01 0x55 0x01 0x69 (Address set successfully)

Trigger measurement

Command:

Header		Address	Length	Cmd	SUM
55	aa	AD	0	01	SUM

Return Value：

None

PS: Trigger one measure. The distance data will be available after 30ms. This command do not return any data. The distance data is stored in the buffer, and the Read Distance command can be applied to get this distance data.

Example:

Command:

0x55 0xaa 0x00 0x01 0x00

Return:

None

Read Distance

Command:

Header		Address	Length	Cmd	SUM
55	aa	ADD	0	02	SUM

Header		Address	Length	Cmd	High Byte	Low Byte	SUM
55	aa	ADD	2	02	H	L	SUM

PS: The command will return the measured distance value. The value consists of two bytes. If the measurement is out of range or unsuccessful, the return data will be “0xFF(H) 0xFF(L)”.

Example:

Command:

0x55 0xaa 0x11 0x00 0x02 0x12(SUM)

Return:

0x55 0xaa 0x11 0x02 0x02 0x01 0x0A 0x11 (Distance is 266 cm)

0x55 0xaa 0x11 0x02 0x02 0xFF 0xFF 0x1F (Out of Range)

Read temperature

Command:

Header		Address	Length	Cmd	SUM
55	aa	ADD	0	03	SUM

Header		Address	Length	Cmd	High Byte	Low Byte	SUM
55	aa	ADD	2	03	H	L	SUM

PS: The command will return the temperature reading. The return temperature reading is using Celsius scale. If the temperature is above 0 Celsius, the first four bits of High will be all 0. If the temperature is below 0 Celsius, the first four bits of High will be all 1. The last 4 bits of High together with the Low bits stands for 12bits temperature. The resolution is 0.1. When the reading is invalid, it returns 0xFF 0xFF

Example:

Command:

0x55 0xaa 0x11 0x00 0x03 0x13(SUM)

Return:

0x55 0xaa 0x11 0x02 0x03 0xF0 0x0A 0x11 (+1 Celsius Degree)

0x55 0xaa 0x11 0x02 0x03 0x00 0x0A 0x20 (-1 Celsius Degree)

0x55 0xaa 0x11 0x02 0x03 0xFF 0xFF 0x20 (Out of Range)

Simple code:

```
#include <SONAR.h>   //used the library of SONAR

// The sonar whose address is 0x11 was named S11, and it was setted as the type of SONAR
SONAR s11=SONAR(0x11);
//SONAR s12(0x12);

void setup() {
    SONAR::init();          //set up some parameters
    delay(100);                    //100 millissecond
    s11.setAddr(0x11);     //set address for S11with 0x11 .
}

//Ttrigger sonar and display the data.
void loop() {
    s11.trigger();          //Send the trigger command to trigger S11
    //s12.trigger();
    delay(SONAR::duration);                  //60 millissecond.
    Serial.println(s11.getDist(),DEC);       //Display the distance S11 received.

    s11.showDat();                           //Display the data S11 received

    //Serial.println(s12.getDist(),DEC);
    Serial.println(s11.getTemp(),DEC);        //Display the temperature S11 received
    //Serial.println(s12.getTemp(),DEC);
    delay(500);
}
```

➢ Sensor Connection

As the sensor uses RS485 interface which can not be connected directly to the MCU, a MAX485 chip will bridge the TTL interface to RS485.

For PC users, either a USB-RS485 or RS232-RS485 converter will bridge the gap.

➢ Sensor Networking

Up to 32 units are able to join a RS485 network. Simply serially connect the sensors uses twisted pair cables.

➢ APC220 Module

➢ Parameters

Transmission Distance : 800m to 1200m
Arduino Wireless Transmission APC220 PC Kits-1 based on APC220

➢ Kit list

1. one APC220 Wireless Transmission module
2. one APC220 USB adaptation shield
3. one antenna

➢ This module used to connect PC port.

How to use the RF module to control Arduino wirelessly? Its principle is similar to a remote control, which has 4 buttons for RF wireless remote control. However, on occasions of data transmission, such a solution becomes less suitable, for example when you want to send PC the data that Arduino collected from light sensors. It is technically known as the wireless data transmission. At present, there are many solutions for wireless data transmission. A very simple way is connecting with the Arduino using APC220 to send data via serical port. Although the data transmission speed may slow (limited by the serial port baud rate) ,it is a simple and pratical way.No wonder that many netizens recommended the inclusion of such Arduino module support.

Manufacturers do not give any datasheet or material to us. Fortunately, some can be found on the network. Meanwhile learn by researching. First , USB adapter from manufacturer seems not to match APC220 because the number of pins is different. Maybe because it has to be compatible with other different products, or at least it is not specially designed for the APC220. USB adapter used CP2102 chip. Download the appropriate drivers in Silicon Laboratories, To downloaded the file cp210x_vcp_win2k_xp_s2k3.zip, unzipped to get an exe file, then instal the driver step by step following the prompts.

After driver installation is complete, insert USB adapter into the PC's USB interface, Windows will be prompted to find new hardware, then finsh installation and configuration accordingly:

See: http://www.emartee.com/product/41854/Arduino-Wireless-Transmission-APC220-PC-Kits-1

➢ Pin Change Interrupt

➢ PinChangeInt Library

This library was inspired by and derived from the PcInt example by "johnboiles", and written by a "jbrisbin" (it seems).

While the PCInt example shows with generality how Pin Change interrupts can be done under Arduino, it is an 'example' effort that sacrifices performance for clarity of implementation.

The PinChangeInt effort runs in the other direction, with clarity sacrificed for the highest performance. To that end, this implementation can handle a 4.5KHz input signal (with a 1 KHz timer running as well) on an 8 MHzATMega328, with significant code in both the timer and pin change interrupt handlers.

Usage

The PinChangeInt library exposes two functions to to the user and provides macros that make it drop-in compatible with the PcInt code.

To attach an interrupt use

PCintPort::attachInterrupt(pin, userFunc, mode)

or

PCattachInterrupt(pin,userFunc,mode)

To detach an interrupt use

PCintPort::detachInterrupt(pin)

or

PCdetachInterrupt(pin)

See: http://arduino.cc/playground/Main/PinChangeInt

➢ The Code

PinChangeIntConfig.h

See: http://arduino.cc/playground/Main/PinChangeInt

PinChangeInt.h

See : http://arduino.cc/playground/Main/PinChangeInt

PinChangeInt Example

This code counts the number of times pin 15 (aka Analog 1) changes state and prints the count when it recives a p on the serial port.

```
1. /*
2. Copyright 2011 Lex.V.Talionis at gmail
3. This program is free software: you can redistribute it and/or modify it under the terms of the GNU
   General Public License as published by the Free Software Foundation, either version 3 of the
   License, or (at your option) any later version.
4. */
5. #include <PinChangeInt.h>
6. #include <PinChangeIntConfig.h>
```

```
7.
8.  #define PIN 15   // the pin we are interested in
9.  byte burp=0;     // a counter to see how many times the pin has changed
10. byte cmd=0;      // a place to put our serial data
11.  void setup() {
12.    Serial.begin(9600);
13.    Serial.print("PinChangeInt test on pin ");
14.    Serial.print(PIN);
15.    Serial.println();
16.    pinMode(PIN, INPUT);      //set the pin to input
17.    digitalWrite(PIN, HIGH); //use the internal pullup resistor
18.    PCintPort::attachInterrupt(PIN, burpcount,RISING); // attach a PinChange Interrupt to our pin on
       the rising edge
19. // (RISING, FALLING and CHANGE all work with this library)
20. // and execute the function burpcount when that pin changes
21.   }
22.  void loop() {
23.    cmd=Serial.read();
24.   if (cmd=='p')
25.   {
26.      Serial.print("burpcount:\t");
27.      Serial.println(burp, DEC);
28.   }
29.    cmd=0;
30. }
31.  void burpcount()
32. {
33.    burp++;
34. }
```

See: http://arduino.cc/playground/Main/PinChangeIntExample

➢ PID Control

➢ What Is PID

From Wikipedia: "A PID controller calculates an 'error' value as the difference between a measured [Input] and a desired setpoint. The controller attempts to minimize the error by adjusting [an Output]."

So, you tell the PID what to measure (the "Input",) Where you want that measurement to be (the "Setpoint",) and the variable to adjust that can make that happen (the "Output".) The PID then adjusts the output trying to make the input equal the setpoint.

For reference, in a car, the Input, Setpoint, and Output would be the speed, desired speed, and gas pedal angle respectively.

Tuning Parameters

The black magic of PID comes in when we talk about HOW it adjusts the Output to drive the Input towards Setpoint. There are 3 Tuning Parameters (or "Tunings"): Kp, Ki & Kd. Adjusting these values will change the way the output is adjusted. Fast? Slow? God-awful? All of these can be achieved depending on the values of Kp, Ki, and Kd.

So what are the "right" tuning values to use? There isn't one right answer. The values that work for one application may not work for another, just as the driving style that works for a truck may not work for a race car. With each new application you will need to try Several Tuning values until you find a set that gives you what you want.

➢ The Library

Using The PID Library has two benefits in my mind

1. There are many ways to write the PID algorithm. A lot of time was spent making the algorithm in this library as solid as any found in industry. If you want read more about this, check out this detailed explanation.

2. When using the library all the PID code is self-contained. This makes your code easier to understand. It also lets you do more complex stuff, like say having 8 PIDs in the same program.

See: http://www.arduino.cc/playground/Code/PIDLibrary

See: http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/

Sample code

```
/********************************************************
 * PID Basic Example  Reading analog input 0 to control analog PWM output 3
********************************************************/
#include <PID_v1.h>
double Setpoint, Input, Output;       //Define Variables we'll be connecting to
PID myPID(&Input, &Output, &Setpoint,2,5,1, DIRECT);  //Specify the links and initial tuning parameters
void setup() {
```

```
  Input = analogRead(0);       //initialize the variables we're linked to
  Setpoint = 100;
  myPID.SetMode(AUTOMATIC); //turn the PID on
}
void loop() {
  Input = analogRead(0);
  myPID.Compute();
  analogWrite(3,Output);
}
```

See: http://www.arduino.cc/playground/Code/PIDLibrary

➢ Servo control Theory

RC servos are comprised of a DC motor mechanically linked to a potentiometer. Pulse-width modulation (PWM) signals sent to the servo are translated into position commands by electronics inside the servo. When the servo is commanded to rotate, the DC motor is powered until the potentiometer reaches the value corresponding to the commanded position.

Figure 1 shows The servo motor's signal wires, there are two bigs and four smalls.following ,we will to understand how the motor works. we can use these wires to control the motor make servo control.

How to link the Servo motor's signal wires: The two wires Sticked together used for drive motor: power: the red one, marked by "+" should be connected to the positive wire from your supply to the screw terminal labeled "VIN" on the Arduino board. *NOTE: Maximum supply voltage cannot exceed 14V DC.*Ground:The black one. marked by "-" should be connected to a ground pin on the Arduino board. Aother four wires Sticked together is used for encoder: the power: marked by "+" should be connected to the 5V pin on the Arduino board . the Ground : marked by "-" should be connected to a ground pin on the Arduino board. The signal pins: Pin A marked "A", should be connected to a digital pin which one you defined on the Arduino board . It used for input command to motor. Pin B:marked "B". should be connected to a digital pin which one you defined on the Arduino board . It used for output the signal of the motor runs.

Pin PWM Mode

Pin	**name**	**Function**
Digital 4	M1	Motor 1 Direction control
Digital 5	E1	Motor 1 PWM control (speed control)
Digital 6	M2	Motor 2 PWM control (speed control)
Digital 7	E2	Motor 2 Direction control

The PWM DC motor control is implemented by manipulating two digital IO pins and two PWM pins. As illustrated in the diagram above, Pin 4,7 are motor direction control pins, Pin 5,6 are motor speed control pins.Then you can used it to set the motor works as you want.

➢ The PWM signal

In a nutshell, PWM is a way of digitally encoding analog signal levels. Through the use of high-resolution counters, the duty cycle of a square wave is modulated to encode a specific analog signal level. The PWM signal is still digital because, at any given instant of time, the full DC supply is either fully on or fully off. The voltage or current source is supplied to the analog load by means of a repeating series of on and off pulses. The on-time is the time during which the DC supply is applied to the load, and the off-time is the period during which that supply is switched off. Given a sufficient bandwidth, any analog value can be encoded with PWM.

Figure 2 shows three different PWM signals. Figure 1a shows a PWM output at a 10% duty cycle. That is, the signal is on for 10% of the period and off the other 90%. Figures 1b and 1c show PWM outputs at 50% and 90% duty cycles, respectively. These three PWM outputs encode three different analog signal values, at 10%, 50%, and 90% of the full strength. If, for example, the supply is 9V and the duty cycle is 10%, a 0.9V analog signal results.

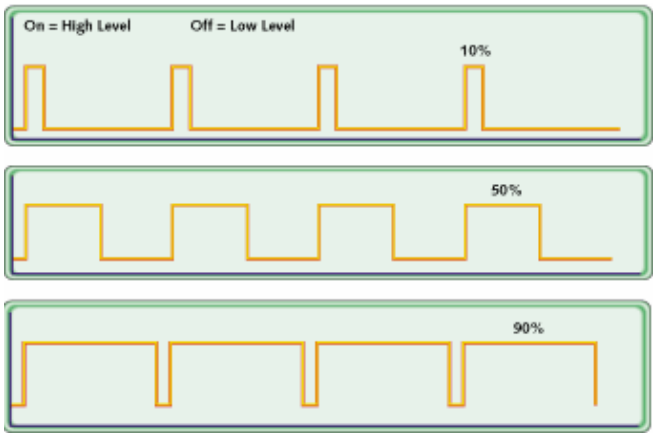
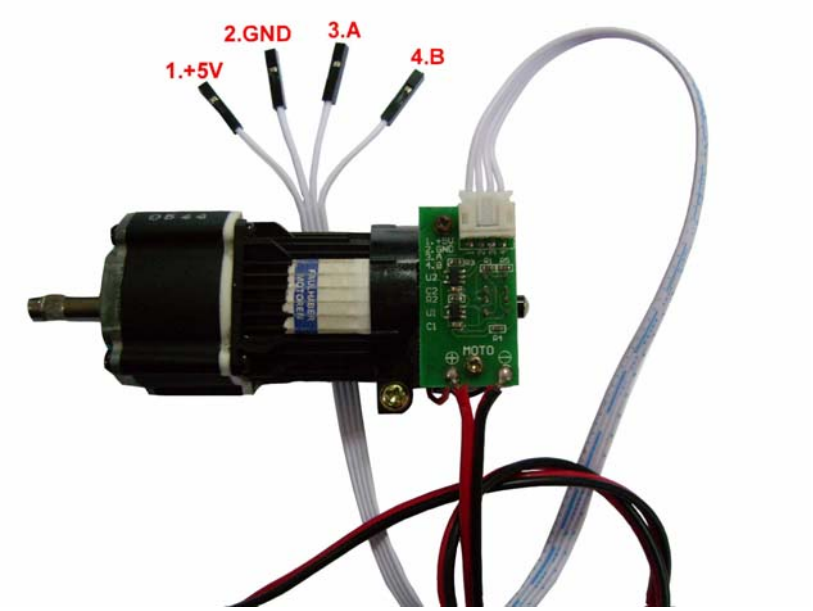


Figure 2. PWM signals of varying duty cycles



motor's encoder wires

See: http://www.netrino.com/Embedded-Systems/How-To/PWM-Pulse-Width-Modulation

➢ Motorwheel

This page describes how to control the built-in Motorwheel.

➢ Motorwheel Class Reference

This document describes how to use the Motor library to control Motors. On the Introduction, you will know how to controls the motor's PWM, direction and speed .

This motorwheel library version 1.1,compatible with maple.

Contents:

struct ISRVars Reference

This section gives a full listing of the capabilities of a struct ISRVars

struct ISRVars

Define a struct named of IRSVars.In the struct ,there are 7 values.

Values:

void (***ISRfunc**)()

A pointer function

volatile long **pulses**

The pulse which got from the interrupt pin, will be used to confirm the direction

volatile unsigned long **pulseStartMicros**

Save the time when the start pulse

volatile unsigned long **pulseEndMicros**

Save the time when the end pulse

volatile unsigned int **speedPPS**

Save the speed of the pulse(pulse per second)

volatile bool **currDirection**

Save the current direction

unsigned char **pinIRQB**

The IRQB pin

➢ Class Motor Reference

This section gives a full listing of the capabilities of a Motor.

Class Motor : public PID

Interface for visit of peripherals . Inherit from the Public PID.

➢ Public functions

Motor(unsigned char _pinPWM,unsigned char _pinDir,

unsigned char _pinIRQ,unsigned char _pinIRQB,

struct ISRVars* _isr)

Construct a new Motor instance.

In your sketch. This will create a Motor object called Motor. You can then use any of its methods; for instance, to control a motor attached to pins, you could write

Parameters:	unsigned char _pinPWM The motor PWM control pin unsigned char _pinDir The motor direction control pin unsigned char _pinIRQ The interrupt pin A unsigned char _pinIRQB The interrupt pin B struct **ISRVars*** _isr The Structure IRSVars's member
See:	**irqISR(y,x)**

void **setupInterrupt**()

Setup a attach interrupt

unsigned char **getPinPWM**() const

Get the motor's PWM control pin number

Return:	PWM control pin

unsigned char **getPinDir**() const

Get the motor's direction control pin number

Return:	Direction pin

unsigned char **getPinIRQ**() const

Get the IRQ Pin number

Return:	IRQ pin

unsigned char **getPinIRQB**() const

Get the IRQB Pin number

Return:	IRQB pin

unsigned int **runPWM**(unsigned int PWM,bool dir,bool saveDir=true);

Set the PWM and direction for Motors .then return the PWM.

Parameters:	unsigned int PWM The PWM set for the Motor Bool dir The direction set for the Motor Bool saveDir A flag to confirm if the direction will be reset
Return:	PWM

unsigned int **getPWM**() const

Get the motor's current pwm

Return:	Speed PWM

unsigned int **advancePWM**(unsigned int PWM)

Set the pwm when the motor run advance

Parameters:	unsigned int PWM The PWM set for Motor
Return:	runPWM(PWM,DIR_ADVANCE)
See:	**Motor:: runPWM**()

unsigned int **backoffPWM**(unsigned int PWM);

Set the pwm when the motor backoff

This will lie within the range specified at **Motor::runPWM**()

Parameters:	unsigned int PWM The PWM set for Motor
See:	**Motor:: runPWM**()

bool **setDesiredDir**(bool dir)

The desired direction set for the motor

This will lie within the range specified at **Motor::getDesiredDir**()

Parameters:	Bool dir The direction set for Motor
See:	**Motor::getDesiredDir**()

bool **getDesiredDir()** const

Get the desired direction

Return:	Desired Direction

bool **reverseDesiredDir**()

Get the reverse desired direction

Return:	Desired Direction

bool **setCurrDir**()

Set a current direction on the basis of the digitalRead(pinIRQB)

Return:	Current Direction If getPinIRQB() was defined false otherwise

bool **getCurrDir**() const

Get the current direction

Return:	Current Direction

unsigned int **getSpeedRPM**() const

Get the speed of the motor (round per minute)

This will lie within the range specified at **SPEEDPPS2SPEEDRPM**().

See:	**SPEEDPPS2SPEEDRPM**().

unsigned int **setSpeedRPM**(int speedRPM,bool dir)

Set the speed and direction for the motor

This will lie within the range specified at **Motor::PIDSetSpeedRPMDesired**() **, Motor::setDesiredDir**(), **Motor::getSpeedRPM**()

Parameters:	int speedRPM The speed set for Motor Bool dir The direction set for Motor
See:	**Motor:: PIDSetSpeedRPMDesired**() **Motor::setDesiredDir**() **Motor::getSpeedRPM**()

void **simpleRegulate**()

Regulate the speed of the Motor on the basis of the direction

bool **PIDSetup**(float kc=KC,float taui=TAUI,float taud=TAUD,unsigned int sampleTime=1000)

Setup the.The class PID use these datas to regulate the speed of Motors

Parameters:	float kc Proportional term float taui Integral term float taud Derivative term unsigned int sampleTime The time the PID work last
Return:	Bool ture

bool **PIDGetStatus()** const

Get the current PID state,to sure wether the PID works

Return:	The value of the pidCtrl

bool **PIDEnable**(float kc=KC,float taui=TAUI,float taud=TAUD,unsigned int sampleTime=1000)

Enable the PID ,make it works

This will lie within the range specified at **PID::PIDSetup**()

Parameters:	Float kc Proportional term Float taui Integral term Float taud Derivative term Unsigded int sampletime The time the PID work last
Return:	pidCtrl equal ture
See:	pidCtrl equal ture

bool **PIDDisable**()

Disable the PID,release it

Return:	**PID::PIDSetup**()

bool **PIDReset**()

Reset the state of PID

This will lie within the range specified at **PID::Reset**()

Return:	False if the PIDGetStatus() return false Ture ortherwise
See:	**PID::Reset**()

bool **PIDRegulate**(bool doRegulate=true)

Regulate the PID ,in order to adjust the speed of the Motor.

This will lie within the range specified at **PID::Compute**().

parameters	Bool doRegulate A bool value
Return:	False if the PIDGetStatus() return false,Ture ortherwise
See:	**PID::Reset()**

unsigned int **PIDSetSpeedRPMDesired**(unsigned int speedRPM)

According to the User's demands ,use the class PID to set the speed of Motor.

This will lie within the range specified at **PID::PIDGetSpeedRPMDesired**()

parameters	Unsigned int speedRPM The speed User want to set
See:	**PID::PIDGetSpeedRPMDesired**()

unsigned int **PIDGetSpeedRPMDesired**() const

Get the desired speed

Return:	speedRPMDesired

void **debugger**() const

Debug to sure if the result is right

int **getSpeedPPS**() const

Get the pulse rate (pulse per second)

Return:	speedPPS

long **getCurrPulse**() const

Get the current Pulse

Return:	Pulses

long **setCurrPulse**(long _pulse)

Set pulse

This will lie within the range specified at **Motor::getCurrPulse**()

parameters	Long _pulse The value want to set
See:	**Motor::getCurrPulse**()

long **resetCurrPulse**()

Reset the current Pulse

This will lie within the range specified at **Motor::SetCurrPulse**()

See:	**Motor::SetCurrPulse**()

Struct ISRVars * isr

Define a Pointer named isr, as the member of the struct ISRVars

➢ Private members

unsigned char **pinPWM**

The PWM pin.

unsigned char **pinDir**

The diretion pin

unsigned char **pinIRQ**

The IRQ pin

unsigned char **pinIRQB**

The IRQB pin

bool **desiredDirection**

The desired direction

unsigned int **speedPWM**

Save the current PWM

int **speedRPMInput**

Save the Motor's current speed.it will be used in class PID

int **speedRPMOutput**

Save the speed of the Motor output.

int **speedRPMDesired**

Save the speed the user want to set

bool **pidCtrl**

The class PID work's mode

Motor()

Construct a new Motor instance.

After this the class Motor's explain is over

➢ Class GearedMotor

Interface for visit of peripherals and it's Inherit from the Public Motor.

➢ Public functions

GearedMotor(unsigned char _pinPWM,unsigned char _pinDir,

unsigned char _pinIRQ,unsigned char _pinIRQB,

struct ISRVars* _isr,

unsigned int _ratio=REDUCTION_RATIO);

Construct a new GearedMotor instance.

In your sketch. This will create a GearedMotor object called GearedMotor. You can then use any of its methods; for instance, to control a Gearedmotor attached to pins, you could write

parameters	Unsigned char _pinPWM
	The PWM control pin
	unsigned char _pinDir
	The direction control pin
	unsigned char _pinIRQ
	The IRQ pin
	unsigned char _pinIRQB
	The IRQB pin
	struct ISRVars* _isr
	A point of the struct IRSVars's member
	unsigned int _ratio
	A variable equal 60

float **getGearedSpeedRPM**() const

Get the Geared speed (round per second)

This will lie within the range specified at **Motor::getSpeedRPM**(),to understand this ,you will to understand **(float)Motor::getSpeedRPM()/_ratio**

See:	**Motor::getSpeedRPM**()

float **setGearedSpeedRPM** (float gearedSpeedRPM,bool dir)

Set the GearedSpeed

This will lie within the range specified at **Motor::setSpeedRPM** ()

parameters	float gearedSpeedRPM
	The value want to set
	bool dir
	The Motor's direction

Return:	gearedSpeedRPM
See:	**Motor::setSpeedRPM** ()

unsigned int **getRatio**() const

Get the ratio the car run at

Return:	_ ratio

unsigned int **setRatio**(unsigned int ratio=REDUCTION_RATIO)

Set the Ratio the car run at

This will lie within the range specified at **Motor::getRatio**()

parameters	unsigned int ratio The value want to set
See:	**Motor::getRatio**()

➢ Private Parameters

unsigned int _**ratio**

To save a value

➢ Class MotorWheel

Interface for visit of peripherals and it's Inherit from the Public Motor.

➢ Public functions

MotorWheel(unsigned char _pinPWM,unsigned char _pinDir,

unsigned char _pinIRQ,unsigned char _pinIRQB,

struct ISRVars* _isr,

unsigned int ratio=REDUCTION_RATIO,unsigned int cirMM=CIRMM)

Construct a new **MotorWheel** instance.

in your sketch. This will create a **MotorWheel** object called **MotorWheel**. You can then use any of its methods; for instance, to control a **MotorWheel** attached to pins, you could write

parameters	Unsigned char _pinPWM The PWM control pin unsigned char _pinDir The direction control pin

	unsigned char _pinIRQ
	The IRQ pin
	unsigned char _pinIRQB
	The IRQB pin
	struct ISRVars* _isr
	A point of the struct IRSVars's member
	unsigned int _ratio
	A variable equal 60
	unsigned int cirMM
	A variable equal 314 mm

unsigned int **getCirMM**() const

Get the Circumference of the wheel

Return:	_cirMM

unsigned int **setCirMM**(unsigned int cirMM=CIRMM);

Set the Circumference of the wheel

This will lie within the range specified at **MotorWheel::getCirMM**();

parameters	unsigned int cirMM
	The value want to set
See:	**MotorWheel::getCirMM**()

unsigned int **getSpeedCMPM**() const

Get the speed(centimeter per minute)

This will lie within the range specified at **GearedMotor::getGearedSpeedRPM**()**.**

See:	**GearedMotor::getGearedSpeedRPM**()

unsigned int **setSpeedCMPM**(unsigned int cm,bool dir)

Set the speed for motor

This will lie within the range specified at **GearedMotor::setGearedSpeedRPM**().

Then you will see **MotorWheel::getspeedCMPM**()

See:	**GearedMotor::setGearedSpeedRPM**()
	MotorWheel::getspeedCMPM()

unsigned int **getSpeedMMPS**() const

Get the speed (millimeter per second)

This will lie within the range specified at **MotorWheel::getspeedCMPM**().

See: **MotorWheel::getspeedCMPM**()

unsigned int **setSpeedMMPS**(unsigned int mm,bool dir)

Set the speed .

This will lie within the range specified at **MotorWheel::setspeedCMPM**() and **MotorWheel::getspeedCMPM**().

See: **MotorWheel::setspeedCMPM**()
MotorWheel::getspeedCMPM()

For example:

```
#include <PinChangeInt.h>
#include <PinChangeIntConfig.h>
#include <PID_Beta6.h>
#include <MotorWheel.h>
#ifndef MICROS_PER_SEC
#define MICROS_PER_SEC 1000000
#endif
irqISR(irq1,isr1);                                  //This will create a MotorWheel object called Wheel1
MotorWheel    wheel1(9,8,6,7,&irq1); // Motor PWM:Pin9, DIR:Pin8, Encoder A:Pin6, B:Pin7
void setup() {
      TCCR1B=TCCR1B&0xf8|0x01;                      // Pin9,Pin10 PWM 31250Hz, Silent PWM
     wheel1.setSpeedMMPS(100,DIR_ADVANCE);     //Set the pwm speed 100 direction
      wheel1.PIDEnable(KC,TAUI,TAUD,10);              // used whewl1 to call the PIDEnable
      Serial.begin(19200);
}
void loop() {
      wheel1.PIDRegulate();                                    //regulate the PID
      if(millis()%500==0) {
            Serial.print("speedRPM> ");
```

```
            Serial.println(wheel1.getSpeedRPM(),DEC);      //display the speed of the MotorWheel
            Serial.print("MMPS    --> ");
            Serial.println(wheel1.getSpeedMMPS(),DEC);    //display the speed of the motor
          //wheel1.debugger();
      }
}
```

➢ R2WD

This page describes how to control the built-in R2WD. It does not describe how the R2WD work on your board. For more information on that, please refer to **R2WD Class Reference.**

*2 mobile wheel drive
*Aluminum alloy body
*Includes ultrasonic sensors and Bumper sensors
*Optional IR sensors and Fall detect sensors
*Still spare mounts for sensors
*DC motors with encoders
*Microcontroller and IO expansion board
*Programmable with c,c++
*Based on Arduino microcontroller

RB004-2WD mobile robot kit

This robot kit provides an economical introduction to the world of robotics.It has 2 drive wheels and a freewheel.It includes a serials of sensors making it aware of the environment:Sonar sensors to detect the obstructions,IR distance measure sensors used as a fall-arrest detector,bumper sensors to make it turn around while run into something in its way.It is based on Arduino microcontroller.Its aluminium alloy body is firm enough to be mounted with extension equipments.

➢ R2WD Class Reference

This documents describes a car with two Motors. On the Introduction, you will know how to use the R2WD library to control the Motors,then to control the car

#include<MotorWheel.h>

Include the header file **MotorWheel.h**

This section gives a full listing of the capabilities of **R2WD**

Class R2WD

Interface for visit of peripherals

RB015_Tracked Mobile Tank Robot Kit

➢ Public functions

R2WD(MotorWheel*wheelLeft, MotorWheel*wheelRight,
unsigned int wheelspanMM=WHEELSPAN)

Construct a new R2WD instance.

In your sketch. This will create a R2WD object called R2WD. You can then use any of its methods; for instance, to control a R2WD attached to pins, you could write

Parameters:	MotorWheel*wheelLeft A point named wheelLeft as the object of MotorWheel,left wheel MotorWheel*wheelRight A point named wheelRight as the object of MotorWhee,right wheel unsigned int wheelspanMM=WHEELSPAN The two wheels' span

unsigned int **getWheelspanMM**() const

Get the wheel span (millimeter)

Return :	Wheel span

unsigned int **setWheelspanMM**(unsigned int wheelspan)

Set the wheel span

This will lie within the range specified at **R2WD::getWheelspanMM**()

Parameters:	Unsigned int wheelspan The value want to set
see :	**R2WD::getWheelspanMM**()

unsigned char **switchMotors**()

Switch the motor to control

This will lie within the range specified at **R2WD::getSwitchMotorsStat**()

see :	**R2WD::getSwitchMotorsStat**()

unsigned char **switchMotorsReset**()

Reset the switch about motor's control

This will lie within the range specified at **R2WD::getSwitchMotorsStat**()

see :	**R2WD::getSwitchMotorsStat**()

unsigned int **setCarStop**()

Set the car stop

This will lie within the range specified at **R2WD::setCarStat**() and **R2WD::setMotorAll**()

see :	**R2WD::setCarStat**() **R2WD::setMotorAll**()

unsigned int **setCarAdvance**(unsigned int speedMMPS=0)

Set car move advance

This will lie within the range specified at **R2WD::setCarStat**() and

R2WD::setcaradvanceBase()

Parameters:	unsigned int speedMMPS The speed of the car moves , initialize it
see :	**R2WD::setCarStat**() **R2WD::setadvanceBase**()

unsigned int **setCarBackoff**(unsigned int speedMMPS=0)

Set car move backoff

This will lie within the range specified at **R2WD::setCarStat**() and

R2WD::setcarbackoffBase()

Parameters:	unsigned int speedMMPS The speed of the car moves , initialize it
see :	**R2WD::setCarStat**() **R2WD::setcarbackoffBase**()

unsigned int **setCarRotateLeft**(unsigned int speedMMPS)

Set car move as rotate Left

This will lie within the range specified at **R2WD::setCarStat**() and

R2WD::setMotorAllBackoff()

Parameters:	unsigned int speedMMPS The speed of the car moves , initialize it
see :	**R2WD::setCarStat**() **R2WD::setMotorAllBackoff**()

unsigned int **setCarRotateRight**(unsigned int speedMMPS)

Set car moves rotate right

This will lie within the range specified at **R2WD::setCarStat**() and

R2WD::setMotorAllAdvance ()

Parameters:	unsigned int speedMMPS The speed of the car moves , initialize it
see :	**R2WD::setCarStat**() **R2WD::setMotorAllAdvance**()

unsigned int **setCarUpperLeft**(unsigned int speedMMPS,unsigned int radiusMM)

Set car moves upper left

This will lie within the range specified at **R2WD::setCarStat**() and **R2WD::setCarArcBace**()

Parameters:	unsigned int speedMMPS The speed of the car moves , initialize it unsigned int radiusMM The radius the car moves Locus
see :	**R2WD::setCarStat**() **R2WD::setCarArcBace**()

unsigned int **setCarLowerLeft**(unsigned int speedMMPS,unsigned int radiusMM)

Set car moves Lower left

This will lie within the range specified at **R2WD::setCarStat**() and **R2WD::setCarArcBace**()

Parameters:	unsigned int speedMMPS The speed of the car moves , initialize it unsigned int radiusMM The radius the car moves Locus
see :	**R2WD::setCarStat**() **R2WD::setCarArcBace**()

unsigned int **setCarUpperRight**(unsigned int speedMMPS,unsigned int radiusMM)

Set car moves Upper right

This will lie within the range specified at **R2WD::setCarStat**() and **R2WD::setCarArcBace**()

Parameters:	unsigned int speedMMPS The speed of the car moves , initialize it

	unsigned int radiusMM The radius of the car moves Locus
see :	**R2WD::setCarStat**() **R2WD::setCarArcBace**()

unsigned int **setCarLowerRight**(unsigned int speedMMPS,unsigned int radiusMM)

Set car moves Lower right

This will lie within the range specified at **R2WD::setCarStat**() and **R2WD::setCarArcBace**()

Parameters:	unsigned int speedMMPS The speed of the car moves , initialize it unsigned int radiusMM The radius of the car moves Locus
see :	**R2WD::setCarStat**() **R2WD::setCarArcBace**()

unsigned int **setCarAdvanceDistance**(unsigned int speedMMPS,unsigned long distance);

Set the distance the car move advance

This will lie within the range specified at **R2WD::setCarAdvance**() and **2WD::setCarStrightdistance**()

Parameters:	unsigned int speedMMPS The speed ofthe car moves , initialize it unsigned long distance The distance want the car move
see :	**R2WD::setCarAdvance**() **R2WD::setCarStrightdistance**()

unsigned int **setCarBackoffDistance**(unsigned int speedMMPS,unsigned long distance);

To set the distance of the car move backoff

This will lie within the range specified at **R2WD::setCarBackoff**() and **R2WD::setCarStrightdistance**()

Parameters:	unsigned int speedMMPS The speed of the car moves , initialize it unsigned long distance The distance want the car move

see :	**R2WD::setCarBackoff**() **R2WD::setCarStrightdistance**()

unsigned int **setCarRotateLeftAngle**(unsigned int speedMMPS,float radian);

Set the angle when the car moves as rotate left

This will lie within the range specified at **R2WD::setCarRotateLeft**()and **R2WD::setCarRotateAngle**()

Parameters:	unsigned int speedMMPS The speed ofthe car moves , initialize it float radian The radian when the car move as rotate left
see :	**R2WD::setCarRotateLeft**() **R2WD::setCarRotateAngle**()

unsigned int **setCarRotateRightAngle**(unsigned int speedMMPS=0,float radian=0);

Set the angle when the car moves rotate right

This will lie within the range specified at **R2WD::setCarRotateRight**()and **R2WD::setCarRotateAngle**()

Parameters:	unsigned int speedMMPS =0 The speed ofthe car moves , initialize it float radian =0 The radian when the car move as rotate right, initialize it with 0
see :	**R2WD::setCarRotateRight**() **R2WD::setCarRotateAngle**()

unsigned int **setCarUpperLeftTime**(unsigned int speedMMPS=0,unsigned int

radiusMM=WHEELSPAN,unsigned long duration=5000,unsigned int uptime=500);

Set period of time for the car moves upper left

This will lie within the range specified at **R2WD::setCarUpperLeft**()and **R2WD::setCarArcTime** ()

Parameters:	unsigned int speedMMPS =0 The speed of the car moves , initialize it unsigned int radiusMM=WHEELSPAN The radiusMM of the car move as upper left, initialize it unsigned long duration=5000

	The time of the car last , initialize it unsigned int uptime=500 The time the car used to stop
see :	**R2WD::setCarUpperLeft**() **R2WD::setCarArcTime** ()

unsigned int **setCarLowerLeftTime**(unsigned int speedMMPS=0,unsigned int radiusMM=WHEELSPAN,unsigned long duration=5000,unsigned int uptime=500);

Set period of time for the car moves Lower left

This will lie within the range specified at **R2WD::setCarLowerLeft**()and **R2WD::setCarArcTime** ()

Parameters:	unsigned int speedMMPS =0 The speed ofthe car moves , initialize it unsigned int radiusMM=WHEELSPAN The radiusMM when the car move as upper left, initialize it unsigned long duration=5000 The time the car last , initialize it unsigned int uptime=500 The time the car used to stop
see :	**R2WD::setCarLowerLeft**() **R2WD::setCarArcTime** ()

unsigned int **setCarUpperRightTime(**unsigned int speedMMPS=0,unsigned int radiusMM=WHEELSPAN,unsigned long duration=5000,unsigned int uptime=500);

Set period of time for the car moves upper right

This will lie within the range specified at **R2WD::setCarUpperRight**()and **R2WD::setCarArcTime** ()

Parameters:	unsigned int speedMMPS =0 The speed ofthe car moves , initialize it unsigned int radiusMM=WHEELSPAN The radiusMM when the car move as upper left, initialize it unsigned long duration=5000 The time the car last , initialize it unsigned int uptime=500

	The time the car used to stop
see :	**R2WD::setCarUpperRight**() **R2WD::setCarArcTime** ()

unsigned int **setCarLowerRightTime**(unsigned int speedMMPS=0,unsigned int radiusMM=WHEELSPAN,unsigned long duration=5000,unsigned int uptime=500);

Set period of time for the car moves Lower Right

This will lie within the range specified at **R2WD::setCarLowerRight**()and **R2WD::setCarArcTime** ()

Parameters:	unsigned int speedMMPS =0 The speed for the car to moves ,initialize it unsigned int radiusMM=WHEELSPAN The radiusMM when the car move as upper left, initialize it unsigned long duration=5000 The time the car last , initialize it unsigned int uptime=500 The time the car used to stop
see :	**R2WD::setCarLowerRight**() **R2WD::setCarArcTime** ()

unsigned int **setCarUpperLeftAngle**(unsigned int speedMMPS=0,unsigned int radiusMM=WHEELSPAN,float radian=0,unsigned int uptime=500);

Set the angle when the car moves as Upper left

This will lie within the range specified at **R2WD::setCarUpperLeft**()and **R2WD::setCarArcAngle**()

Parameters:	unsigned int speedMMPS The speed for the car to moves Unsigned int radian The radian when the car move upper left
see :	**R2WD::setCarUpperLeft**() **R2WD::setCarArcAngle**()

unsigned int **setCarLowerLeftAngle**(unsigned int speedMMPS=0,unsigned int radiusMM=WHEELSPAN,float radian=0,unsigned int uptime=500);

Set the angle when the car moves Upper left

This will lie within the range specified at **R2WD::setCarLowerLeft**()and **R2WD::setCarArcAngle**()

Parameters:	unsigned int speedMMPS The speed for the car to moves Unsigned int radian The radian when the car moves lower left
see :	**R2WD::setCarLowerLeft**() **R2WD::setCarArcAngle**()

unsigned int **setCarUpperRightAngle**(unsigned int speedMMPS=0,unsigned int radiusMM=WHEELSPAN,float radian=0,unsigned int uptime=500);

Set the angle when the car moves Upper right

This will lie within the range specified at **R2WD::setCarUpperRight**() and **R2WD::setCarArcAngle**()

Parameters:	unsigned int speedMMPS The speed for the car to moves Unsigned int radian The radian when the car make upper right moves
see :	**R2WD::setCarUpperRight**() **R2WD::setCarArcAngle**()

unsigned int **setCarLowerRightAngle**(unsigned int speedMMPS=0,unsigned int radiusMM=WHEELSPAN,float radian=0,unsigned int uptime=500);

Set the angle when the car moves Upper Right

This will lie within the range specified at **R2WD::setCarLowerRight**()and **R2WD::setCarArcAngle**()

Parameters:	unsigned int speedMMPS The speed for the car to moves Unsigned int radian The radian when the car make lower Right moves
see :	**R2WD::setCarLowerRight**() **R2WD::setCarArcAngle**()

unsigned int **wheelLeftSetSpeedMMPS**(unsigned int speedMMPS=0,bool dir=DIR_ADVANCE);

Set the speed for Left wheel

This will lie within the range specified at **MotorWheel::setSpeedMMPS** ()

Parameters:	unsigned int speedMMPS The speed for the car to moves bool dir=DIR_ADVANCE The direction for the left wheel
see :	**MotorWheel::setSpeedMMPS** ()

unsigned int **wheelLeftGetSpeedMMPS**() const;

Get the speed of the left wheel

This will lie within the range specified at **MotorWheel::getSpeedMMPS** ()

see :	**MotorWheel::getSpeedMMPS** ()

unsigned int **wheelRightSetSpeedMMPS**(unsigned int speedMMPS=0,bool dir=DIR_ADVANCE);

Set the speed for right wheel

This will lie within the range specified at **MotorWheel::setSpeedMMPS** ()

Parameters:	unsigned int speedMMPS The speed for the car to moves bool dir=DIR_ADVANCE The direction for the right wheel
see :	**MotorWheel::setSpeedMMPS** ()

unsigned int **wheelRightGetSpeedMMPS**() const;

Get the speed of the right wheel

This will lie within the range specified at **MotorWheel::getSpeedMMPS** ()

see :	**MotorWheel::getSpeedMMPS** ()

bool **PIDEnable**(float kc=KC,float taui=TAUI,float taud=TAUD,unsigned int interval=1000);

Call the PID,make it work for the car

This will lie within the range specified at **MotorWheel::PIDEnable**()

Parameters:	Float kc Proportional term,initialize it

	Float taui
	Integral term
	Float taud
	Derivative term
	Unsigded int interval
	The time the PID work last
see :	**MotorWheel::PIDEnable**()

bool **PIDRegulate**()

Regulate the PID ,in order to adjust the speed of the Motor.

This will lie within the range specified at **MotorWheel:: PIDRegulate**()

see :	**MotorWheel:: PIDRegulate**()

void **delayMS**(unsigned long ms=100, bool debug=false)

Last time for the car work as the same action

In the function,every 10 milliseconds,it will call the function PIDRegulate once

Parameters:	unsigned long ms=100
	The time the action last
	bool debug=false
	A flag

unsigned int **getCarSpeedMMPS**() const

Get the car's speed

This will lie within the range specified at **R2WD:: wheelLeftGetSpeedMMPS**() and **R2WD::wheelRightGetSpeedMMPS**()

see :	**R2WD:: wheelLeftGetSpeedMMPS**()
	R2WD:: wheelRightGetSpeedMMPS()

unsigned int **setCarSpeedMMPS**(unsigned int speedMMPS=0,unsigned int ms=1000);

Set the car's speed, when the car's state was one of the following :

STAT_ADVANCE

STAT_BACKOFF

STAT_ROTATELEFT

STAT_ROTATERIGHT

This will lie within the range specified at **R2WD::GetCarSpeedMMPS**()

Parameters:	unsigned int speedMMPS=0 The speed for the car to moves ,initialize it unsigned int ms=1000 The time the car's speed changed from 0 to speedMMPS used
see :	**R2WD::getCarSpeedMMPS**()

unsigned int **setCarSpeedMMPSArc**(unsigned int speedMMPS=0,unsigned int radiusMM=WHEELSPAN,unsigned int ms=1000)

Set the car's speed, when the car's state was one of the following :

STAT_UPPERLEFT

STAT_LOWERLEFT

STAT_LOWERRIGHT

STAT_UPPERRIGHT

This will lie within the range specified at **R2WD::getCarSpeedMMPS**()

Parameters:	unsigned int speedMMPS=0 The speed for the car to moves ,initialize it unsigned int radiusMM=WHEELSPAN The radius the car moves unsigned int ms=1000 The time the car's speed changed from 0 to speedMMPS used
see :	**R2WD::getCarSpeedMMPS**()

unsigned int **setCarSlow2Stop**(unsigned int ms=1000)

Stop the car slowly

This will lie within the range specified at **R2WD::setCarSpeedMMPS**() or **R2WD::setCarSpeedMMPSArc**()

Parameters:	unsigned int ms=1000 The time stop the car used,initialize it
see :	**R2WD::getCarSpeedMMPS**()

void **debugger**(bool wheelLeftDebug=true,bool wheelRightDebug=true) const;

Debug the all wheel's speed

Car _state enum

Used to configure the behavior of a car.

Note that not all car can be configured in every state.

Variables:

STAT_UNKNOWN

The state of the car unknown

STAT_STOP

The car's state is stop

STAT_ADVANCE

The car's state is moves advance

STAT_BACKOFF

The car's state is get backoff

STAT_ROTATELEFT

The car's state is moves rotateleft

STAT_ROTATERIGHT

The car's state is moves rotateright

STAT_UPPERLEFT

The car's state is moves upperleft

STAT_LOWERLEFT

The car's state is moves lowerleft

STAT_LOWERRIGHT

The car's state is moves lowerright

STAT_UPPERRIGHT

The car's state is moves uppperright

unsigned char **getCarStat**() const

Get the car current state

return : The car's state

Motor _state enum

Used to configure the behavior of a motor.

Note that not all motors can be configured in every state.

Variables:

MOTORS_FB

The switchmotorstat is FB

MOTORS_BF

The switchmotorstat is BF

unsigned char **getSwitchMotorsStat**() const

Get the state of the Motor

return : The motor's state

unsigned int **getRadiusMM**() const

Get the radius the car moves

If the car state was rotateleft or rotateright.

This will lie within the range specified at **R2WD::getWheelspanMM**

return : radius

➢ Private parameters

MotorWheel* _**wheelLeft**

A point named wheelLeft as the object of MotorWheel

MotorWheel* _**wheelRight**

A point named wheelRight as the object of MotorWheel

unsigned int _**wheelspanMM**

Save a data of the span, will set to the wheel

unsigned char _**carStat**

Save the state of the car

unsigned char **setCarStat**(unsigned char stat)

Set the Car's state

Parameters: unsigned char stat

	the state want to set
return :	Carstate if the stat in the range of the want STAT_UNKNOWN otherwise

unsigned char _**switchMotorsStat**

Save the state of the Motor,prepare for switch motors

unsigned char **setSwitchMotorsStat**(unsigned char switchMotorsStat)

Set the Motors' state

This will lie within the range specified at **R2WD::getSwitchMotorsStat**()

Parameters:	unsigned char switchMotorsStat The state want to set
See:	**R2WD::getSwitchMotorsStat**()

unsigned int _**radiusMM**

Save the data of the radius

unsigned int **setRadiusMM**(unsigned int radiusMM)

Set the radius for the car moves

This will lie within the range specified at **R2WD::getRadiusMM**()

Parameters:	unsigned int radiusMM The radius want to set
See:	**R2WD::getSwitchMotorsStat**()

R2WD()

Construct a new R2DW instance.

unsigned int **setMotorAll**(unsigned int speedMMPS=0,bool dir=DIR_ADVANCE)

Set all motors as the same speed and direction

This will lie within the range specified at **R2WD:: wheelLeftSetSpeedMMPS**()and

R2WD::wheelRightSetSpeedMMPS()

Parameters:	unsigned int speedMMPS=0 The speed set the motors run,initialize it

	bool dir=DIR_ADVANCE The direction set the motors run,initialize it
See:	**R2WD:: wheelLeftSetSpeedMMPS**() **R2WD:: wheelRightSetSpeedMMPS**()

unsigned int **setMotorAllStop**()

Set all Motors stop

This will lie within the range specified at **R2WD::setMotorAll**()

See:	**R2WD::setMotorAll**()

unsigned int **setMotorAllAdvance**(unsigned int speedMMPS=0)

Set all motors run advance

This will lie within the range specified at **R2WD::setMotorAll**()

Parameters:	unsigned int speedMMPS=0 The speed set the motors run,initialize it
See:	**R2WD::setMotorAll**()

unsigned int **setMotorAllBackoff**(unsigned int speedMMPS=0)

Set all motors run backoff

This will lie within the range specified at **R2WD::setMotorAll**()

Parameters:	unsigned int speedMMPS=0 The speed set the motors run,initialize it
See:	**R2WD::setMotorAll**()

unsigned int **setCarAdvanceBase**(unsigned int speedMMPSL=0,unsigned int speedMMPSR=0)

Set car moves advance

This will lie within the range specified at **R2WD:: wheelLeftSetSpeedMMPS**() and

R2WD::wheelRightSetSpeedMMPS() and **R2WD::getCarSpeedMMPS**()

Parameters:	unsigned int speedMMPSL=0 The speed set to the left motor unsigned int speedMMPSR=0 The speed set to the right motor

See:	**R2WD:: wheelLeftSetSpeedMMPS**() **R2WD:: wheelRightSetSpeedMMPS**() **R2WD::getCarSpeedMMPS**()

unsigned int **setCarBackoffBase**(unsigned int speedMMPSL=0,unsigned int speedMMPSR=0)

Set car moves backoff

This will lie within the range specified at **R2WD:: wheelLeftSetSpeedMMPS**() and

R2WD::wheelRightSetSpeedMMPS() and **R2WD::getCarSpeedMMPS**()

Parameters:	unsigned int speedMMPSL=0 The speed set to the left motor unsigned int speedMMPSR=0 The speed set to the right motor
See:	**R2WD:: wheelLeftSetSpeedMMPS**() **R2WD:: wheelRightSetSpeedMMPS**() **R2WD::getCarSpeedMMPS**()

unsigned int **setCarRotateAngle**(unsigned int speedMMPS=0,float radian=0)

Set the angle when the car moves rotate

This will lie within the range specified at **R2WD::getWheelspanMM**()

Parameters:	unsigned int speedMMPS=0 The speed set to the motors,initialize it float radian=0 The radian set to the car moves,initialize it
Return:	timeMS,the time the car moves
See:	**R2WD::getWheelspanMM**()

unsigned int **setCarStraightDistance**(unsigned int speedMMPS=0,unsigned long distance=0);

Set the straight distance the car moves

Parameters:	unsigned int speedMMPS=0 The speed set the motors at,initialize it unsigned long distance=0 The distance set the car moves,initialize it

Return:	timeMS,the time the car moves

unsigned int **setCarArcBase**(unsigned int speedMMPS=0,unsigned int radiusMM=WHEELSPAN)

Set the Arc the car moves

The car have arc is on the basis of the two wheel have diffrent speed or direction

This will lie within the range specified at **R2WD::setCarBackoffBase**() and

R2WD::setCarAdvanceBase()

Parameters:	unsigned int speedMMPS=0 The speed set the motors run at,initialize it int radiusMM=WHEELSPAN The radius set the car moves ,initialize it
Return:	timeMS,the time the car moves

unsigned int **setCarArcTime**(unsigned int speedMMPS=0,unsigned int radiusMM=WHEELSPAN,

unsigned long duration=5000,unsigned int uptime=500)

Set the time the car moves as Arc

Parameters:	unsigned int speedMMPS=0 The speed set the car moves unsigned int radiusMM=WHEELSPAN The radius the car moves unsigned long duration=5000 The time the car last unsigned int uptime=500 The time the car used to stop
Return:	The time the car used to moves and stop

unsigned int **setCarArcAngle**(unsigned int speedMMPS=0,unsigned int radiusMM=WHEELSPAN,

float radian=0,unsigned int uptime=500)

Set the Arc angle the car moves

Parameters:	unsigned int speedMMPS=0 The speed set the car moves unsigned int radiusMM=WHEELSPAN

	The radius the car moves float radian=0 The radian the car moves unsigned int uptime=500 The time the car used to stop
Return:	The time the car used to moves and stop

void **demoActions**(unsigned int speedMMPS=100,unsigned int duration=5000,unsigned int uptime=500,bool debug=false)

A function for the car demo action

```
void R2WD::demoActions(unsigned int speedMMPS,unsigned int duration,unsigned int uptime,bool debug) {
    unsigned int (R2WD::*carAction[])(unsigned int speedMMPS)={
        &R2WD::setCarAdvance, // set car to moves advance
        &R2WD::setCarBackoff,   //set car to moves back off
        &R2WD::setCarRotateLeft, //set car to moves as rotate left
        &R2WD::setCarRotateRight, ////set car to moves as rotate right
    };
    unsigned int (R2WD::*carAction2[])(unsigned int speedMMPS,unsigned int radiusMM)={
        &R2WD::setCarUpperLeft,   //set the car moves as upper left
        &R2WD::setCarLowerLeft,    //set the car moves as Lower left
        &R2WD::setCarUpperRight,   //set the car moves as upper right
        &R2WD::setCarLowerRight,   //set the car moves as lower right
    };
    for(int i=0;i<8;++i) {                    //the demo have 8 actions
        if(i<4) {                                 //the first four action
            (this->*carAction[i])(0);
            setCarSpeedMMPS(speedMMPS,uptime);      // speedMMPS=100 set the car's speed is 100
        } else {                                  //the last four action
            (this->*carAction2[i-4])(0,500);
```

```
                setCarSpeedMMPSArc(speedMMPS,getRadiusMM(),uptime);
            }
            delayMS(duration,debug);    //duration=5000    the car moves this action willl last 5000
milliseconds
            setCarSlow2Stop(uptime);    //uptime=500 set the car stop in 500 milliseconds
        }
}
```

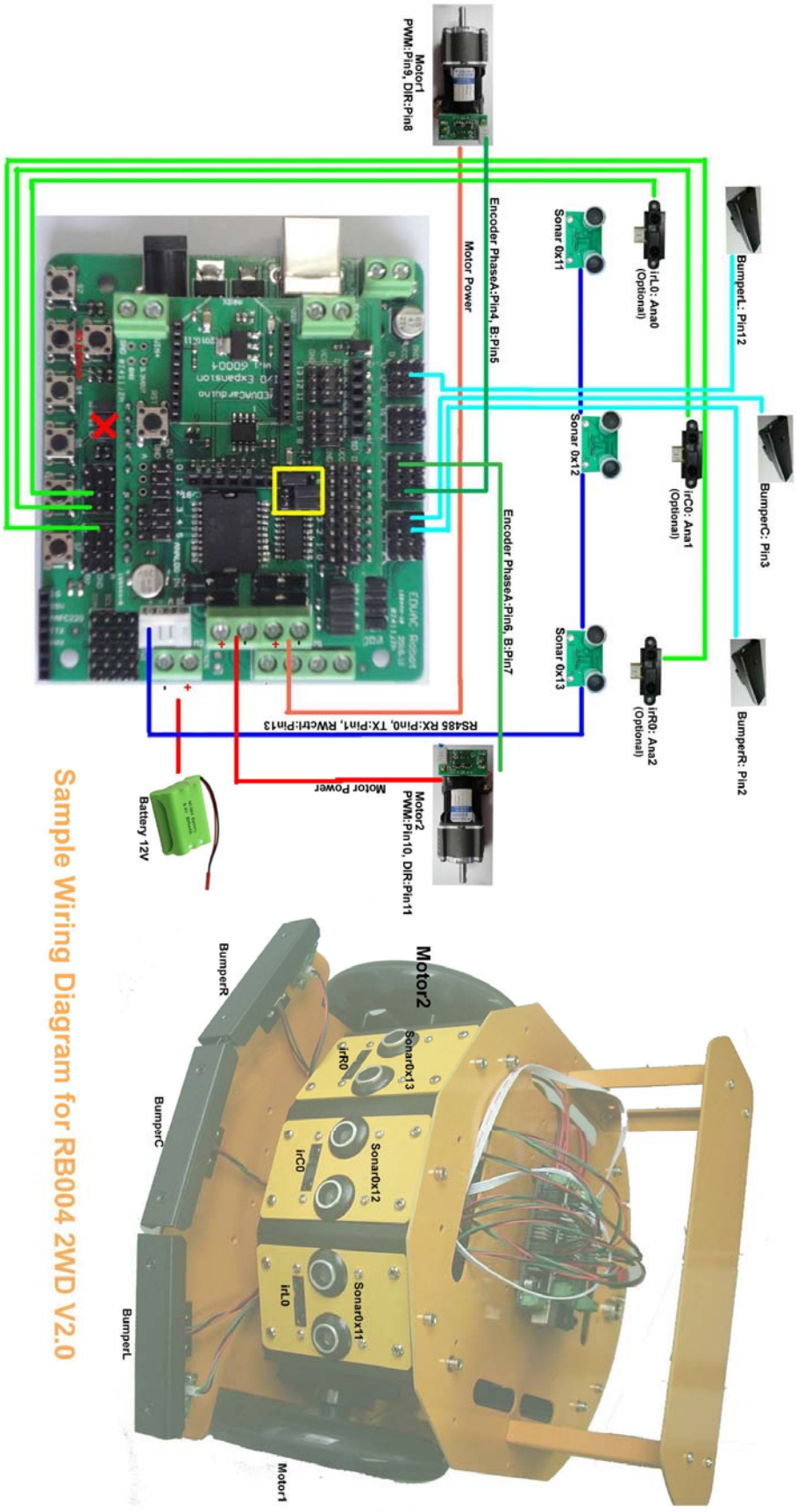
➢ R2WD_test

Here's an example ,we use it to test a car with two wheels.after this ,you will More thorough understanding of the library

Simple code:

```
#include <MotorWheel.h>
#include <R2WD.h>

#include <PID_Beta6.h>
#include <PinChangeInt.h>
#include <PinChangeIntConfig.h>      // Include the header files

/*

      Wheel1 ||-----|| Wheel2

 */
irqISR(irq1,isr1);                    // Intterrupt function.on the basis of the pulse     ,work for wheel1
MotorWheel wheel1(9,8,4,5,&irq1,REDUCTION_RATIO,int(144*PI));
                                                //This will create a MotorWheel object called Wheel1
                                                //Motor PWM:Pin9, DIR:Pin8, Encoder A:Pin4, B:Pin5

irqISR(irq2,isr2);
```

```
MotorWheel wheel2(10,11,6,7,&irq2,REDUCTION_RATIO,int(144*PI));


R2WD _2WD(&wheel1,&wheel2,WHEELSPAN);
                                        // This will create a R2WD object called R2WD. You
                                       // can then use any of its methods; for instance, to
                                      // control a R2WD attached to pins, you could write


void setup() {
      //TCCR0B=TCCR0B&0xf8|0x01;        //   warning!! it will change millis()
      TCCR1B=TCCR1B&0xf8|0x01;          //   Pin9,Pin10 PWM 31250Hz
      //TCCR2B=TCCR2B&0xf8|0x01;       //   Pin3,Pin11 PWM 31250Hz


      _2WD.PIDEnable(0.26,0.01,0,10);   // Enable PID
}


void loop() {
      _2WD.demoActions(100,5000);        // Call the demoActions from the Class R2WD
      /*
            _2WD.setCarAdvance(0);
            _2WD.setCarSpeedMMPS(100,500);
            _2WD.delayMS(5000);
            _2WD.setCarSlow2Stop(500);


            _2WD.setCarBackoff(0);
            _2WD.setCarSpeedMMPS(100,500);
            _2WD.delayMS(5000);
            _2WD.setCarSlow2Stop(500);
       */
}
```



Sample Wiring Diagram for RB004 2WD V2.0

➢ 2WD platform with 3 SONAR

Look the above figure of simple Wiring Diagram for RB004 2WD V2.0. this code is matched for it

RB004_2WD_PID_3SONAR_3IR code

```
#include <MotorWheel.h>
#include <Omni3WD.h>
#include <Omni4WD.h>
#include <R2WD.h>
#include <PID_Beta6.h>
#include <PinChangeInt.h>
#include <PinChangeIntConfig.h>

#include <SONAR.h>                // Include the header files

/*****************************************/
// SONAR

SONAR sonar11(0x11),sonar12(0x12),sonar13(0x13);     // Software initialization
                                                      //SONAR is be defined in <SONAR.h> as a class

unsigned short distBuf[3];                  // Used to save the data of the 3 sonars return;
void sonarsUpdate() {
    static unsigned char sonarCurr=1;       // A variable save data used to flag the sonar's state
    if(sonarCurr==3) sonarCurr=1;
    else ++sonarCurr;
    if(sonarCurr==1) {                      // The conditions is ture?
        distBuf[1]=sonar12.getDist();       // Get the value of distance from sonar12
        sonar12.trigger();                  // Trigger sonar12
    } else if(sonarCurr==2) {
        distBuf[2]=sonar13.getDist();     // Get the value of distance from sonar13
        sonar13.trigger();                  // Trigger sonar13
    } else {
        distBuf[0]=sonar11.getDist();     // Ger the value of distance from sonar11
        sonar11.trigger();                  // Trigger sonar11
    }
}

/*********************************************/
// Infrared Sensor
```

```
unsigned char irL0_pin=0;       // Set Analog pin 0 as the left Infrared Sensor signal pin
unsigned char irC0_pin=1;
unsigned char irR0_pin=2;        // Set Analog pin 2 as the right Infrared Sensor signal pin

int ir_distance(unsigned char ir) {
      int val=analogRead(ir);            // Read the data from the Infrared Sensors
      return (6762/(val-9))-4;             // Change the data
}

/*********************************************/
// bumper
unsigned char bumperL_pin=12;       // Set pin 12 as the left Bumper's Sensor signal pin
unsigned char bumperC_pin=3;        // Set pin 3 as the front Bumper's Sensor signal pin
unsigned char bumperR_pin=2;       // Set pin 2 as the right Bumper's Sensor signal pin

/ *********************************************/

irqISR(irq1,isr1);              // Intterrupt function.on the basis of the pulse, work for wheel1
MotorWheel wheel1(9,8,4,5,&irq1,REDUCTION_RATIO,int(144*PI));
                                          //This will create a MotorWheel object called Wheel1
                                          //Motor PWM:Pin9, DIR:Pin8, Encoder A:Pin4, B:Pin5
irqISR(irq2,isr2);
MotorWheel wheel2(10,11,6,7,&irq2,REDUCTION_RATIO,int(144*PI));

R2WD _2WD(&wheel1,&wheel2,WHEELSPAN);
unsigned int speedMMPS=80;

void setup() {
      //TCCR0B=TCCR0B&0xf8|0x01;       // warning!! it will change millis()
      TCCR1B=TCCR1B&0xf8|0x01;       // Pin9,Pin10 PWM 31250Hz
      //TCCR2B=TCCR2B&0xf8|0x01;       // Pin3,Pin11 PWM 31250Hz

      SONAR::init(13);       // Pin13 as RW Control

      _2WD.PIDEnable(0.26,0.02,0,10);             // Enable PID
}
```

```
/*
void loop() {
      _2WD.demoActions(80,5000);     //Call the demoActions from the class 2WD
}
 */

void loop() {
            boolean bumperL=!digitalRead(bumperL_pin);     // a flag to sure if the Left have someting
            boolean bumperC=!digitalRead(bumperC_pin);
            boolean bumperR=!digitalRead(bumperR_pin);

       int irL0=ir_distance(irL0_pin);     // A variable to save the data of the left Infrared Sensor return
       int irC0=ir_distance(irC0_pin);
        int irR0=ir_distance(irR0_pin);

            static unsigned long currMillis=0;
            if(millis()-currMillis>SONAR::duration) {   //every 60ms call sonarUpdate() once
                currMillis=millis();
                sonarsUpdate();
            }

            if(bumperL || bumperC || bumperR) {       // If the car hit something
               _2WD.setCarBackoff(speedMMPS);    // Set car backoff at the speed of speedMMPS
               _2WD.delayMS(300);                        // last 300 ms
               if(bumperL || bumperC) _2WD.setCarRotateRight(speedMMPS); // // back off and turn right
               else _2WD.setCarRotateLeft(speedMMPS);          // back off and turn left
               _2WD.delayMS(300);
            } else if(0<irL0 && irL0<30 || 0<irC0 && irC0<40 || 0<distBuf[0] && distBuf[0]<30 || 0<distBuf[1]
&& distBuf[1]<40) {                                          // If any of these conditions was ture?
               _2WD.setCarRotateRight(speedMMPS);    // Set car rotateright
            } else if(0<irR0 && irR0<30 || 0<distBuf[2] && distBuf[2]<30) {
               _2WD.setCarRotateLeft(speedMMPS);
            } else {                                          // The is nothing around the car
               _2WD.setCarAdvance(speedMMPS); // Set car move advance at the speed of speedPPMS
            }
            _2WD.PIDRegulate();                          // PID regulate the speed
}
```

Omni Wheel

Robots employing omni wheel are capable of moving in any direction by changing velocity and direction of each wheel without changing its orientation.As there are small rollers around the circumference of omni wheels which are perpendicular to the rolling direction making them capable of sliding laterally.

Omni wheel

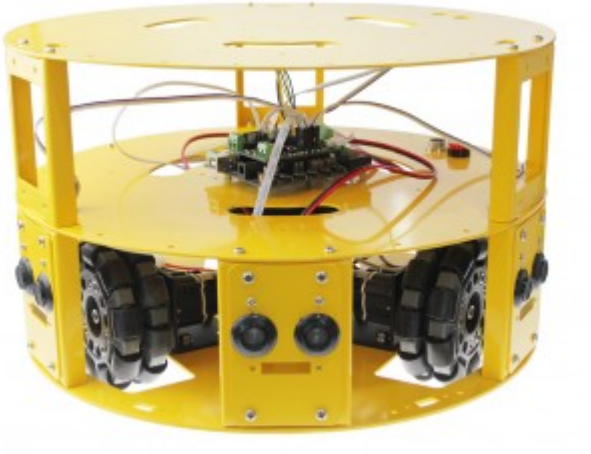
Mecanum wheel

Mecanum wheel

Mecanum wheel is a conventional wheel with a series of rollers attached to its circumference.these rollers have an axis of ratation at 45° to the plane of wheel. A mecanum wheel robot usually is four-wheeled.,the vehicle is stable and can be made to move in any direction and turn by varying the speed and direction of each wheel.Moving all four wheels in the same direction causes forward or backward movement,running the wheels on one side in the opposite direction to those on the orther side causes rotation of the vehicle,and running the wheels on one diagonal in the opposite direction to those on the other diagonal cause sideways movement.Combined motiond motion is also possible.

➢ Omni3WD

This page describes how to control the built-in Omni3WD. It does not describe how the Omni3WD work on your board. For more information on that, Please refer to **Omni3WD Class Reference**.



RB006_3WD omni wheel mobile kit

*3WD 100mm Mecanum wheel
*Aluminum alloy body
*Includes ultrasonic sensors and fall detect sensors
*Optional IR sensors
*With microcontroller and IO expansion board
*DC motors with encoder
*Adequate space for custom components
*Programmable with C,C++
*Based on Arduino microcontroller

*3WD 100mm Mecanum wheel
*3 plate levels
*Aluminum alloy body
*Includes ultrasonic sensors and optional IR
*DC motors with encoder
*With microcontroller and IO expansion board
*Programmable with C,C++
*Based on Arduino microcontroller

RB013_3WD omni wheel mobile kit

RB003_3WD omni wheel mobile kit

*Includes ultrasonic sensors and optional IR
*3WD 100mm Mecanum wheel
*Compact size
*Aluminum alloy body
*DC motors with encoder
*Programmable with C,C++
*Based on Arduino microcontroller

This is a 3 wheels drive mobile robot utilizing omni wheels. It's capable of moving in any directions by changing the velocity and direction of each wheel without changing its orientation. It includes microcontroller, IO expansion board ,DC motor with encoder, ultrasonic sensors and optional IR. With pre-drilled screw holes it can be easily extended.

➢ Omni3WD Class Reference

This document describes a car with three Motors. On the Introduction, you will know how to use the

Omni3WD library to control the Motors,then to control the car

#include<MotorWheel.h>

Include the header file **MotorWheel.h**

This section gives a full listing of the capabilities of **Omni3WD**

Class Omni3WD

Interface for visit of peripherals

➢ Public functions

Omni3WD (MotorWheel* wheelBack,MotorWheel* wheelRight,MotorWheel* wheelLeft)

Construct a new Omni3WD instance.

in your sketch. This will create a Omni3WD object called Omni3WD. You can then use any of its methods;

for instance, to control a Omni3WD attached to pins, you could write

Parameters:	MotorWheel* wheelBack A point named wheelBack as the object of MotorWheel,back wheel MotorWheel*wheelLeft A point named wheelLeft as the object of MotorWheel,left wheel MotorWheel*wheelRight A point named wheelRight as the object of MotorWhee,right wheel

unsigned char **switchMotorsLeft**()

Rotate left to choose the motors

This will lie within the range specified at **Omni3WD::getSwitchMotorsStat**()

See: **Omni3WD::getSwitchMotorsStat**()

unsigned char **switchMotorsRight**()

Rotate right to choose the motors

This will lie within the range specified at **Omni3WD::getSwitchMotorsStat**()

See: **Omni3WD::getSwitchMotorsStat**()

unsigned char **switchMotorsReset**()

Reset the rotate direction to choose the motor

This will lie within the range specified at **Omni3WD::getSwitchMotorsStat**()

See: **Omni3WD::getSwitchMotorsStat**()

unsigned int **setMotorAll**(unsigned int speedMMPS=0,bool dir=DIR_ADVANCE)

Set all the motors' speed and direction

This will lie within the range specified at **Omni3WD::wheelBackSetSpeedMMPS**() and

Omni3WD::wheelRightSetSpeedMMPS() and **Omni3WD::wheelLeftSetSpeedMMPS**()

Parameters: unsigned int speedMMPS=0

	The speed for the motor to run,initialize it. bool dir=DIR_ADVANCE The direction for the motor to run
See:	**Omni3WD::wheelBackSetSpeedMMPS**() **Omni3WD::wheelRightSetSpeedMMPS**() **Omni3WD::wheelLeftSetSpeedMMPS**()

unsigned int **setMotorAllStop**()

Stop all Motors

This will lie within the range specified at **Omni3WD::setMotorAll**()

See:	**Omni3WD::setMotorAll**()

unsigned int **setMotorAllAdvance**(unsigned int speedMMPS=0)

Set all the motors run forward

This will lie within the range specified at **Omni3WD::setMotorAll**()

Parameters:	unsigned int speedMMPS=0 The speed for the motor to run,initialize it.
See:	**Omni3WD::setMotorAll**()

unsigned int **setMotorAllBackoff**(unsigned int speedMMPS=0)

Set all the motors run Reverse

This will lie within the range specified at **Omni3WD::setMotorAll**()

Parameters:	unsigned int speedMMPS=0 The speed for the motor to run,initialize it.
See:	**Omni3WD::setMotorAll**()

unsigned int **setCarStop**()

Stop the car

This will lie within the range specified at **Omni3WD::setMotorAll**() and **Omni3WD::setCarstat**()

See:	**Omni3WD::setMotorAll**() **Omni3WD::setCarstat**()

unsigned int **setCarAdvance**(unsigned int speedMMPS=0)

Set the car moves forward

Because the car have three wheels ,so the car moves forward ,the wheels will have different state.

This will lie within the range specified at **Omni3WD::setCarstat**() and

Omni3WD::wheelBackSetSpeedMMPS() and **Omni3WD::wheelRightSetSpeedMMPS**() and **Omni3WD::wheelLeftSetSpeedMMPS**()

Parameters:	unsigned int speedMMPS=0 The speed for the car moves,initialize it.
See:	**Omni3WD::setCarstat**() **Omni3WD::wheelBackSetSpeedMMPS**() **Omni3WD::wheelRightSetSpeedMMPS**() **Omni3WD::wheelLeftSetSpeedMMPS**()

unsigned int **setCarBackoff**(unsigned int speedMMPS=0)

Set the car moves Reverse

This will lie within the range specified at **Omni3WD::setCarstat**() and

Omni3WD::wheelBackSetSpeedMMPS() and **Omni3WD::wheelRightSetSpeedMMPS**() and **Omni3WD::wheelLeftSetSpeedMMPS**()

Parameters:	unsigned int speedMMPS=0 The speed for the car moves,initialize it.
See:	**Omni3WD::setCarstat**() **Omni3WD::wheelBackSetSpeedMMPS**() **Omni3WD::wheelRightSetSpeedMMPS**() **Omni3WD::wheelLeftSetSpeedMMPS**()

unsigned int **setCarLeft**(unsigned int speedMMPS=0)

Set the car turn Left

This will lie within the range specified at **Omni3WD::setCarstat**() and

Omni3WD::wheelBackSetSpeedMMPS() and **Omni3WD::wheelRightSetSpeedMMPS**() and **Omni3WD::wheelLeftSetSpeedMMPS**()

Parameters:	unsigned int speedMMPS=0 The speed for the car moves,initialize it.
See:	**Omni3WD::setCarstat**()

	Omni3WD::wheelBackSetSpeedMMPS() **Omni3WD::wheelRightSetSpeedMMPS**() **Omni3WD::wheelLeftSetSpeedMMPS**()

unsigned int **setCarRight**(unsigned int speedMMPS=0)

Set the car turn right

This will lie within the range specified at **Omni3WD::setCarstat**() and

Omni3WD::wheelBackSetSpeedMMPS() and **Omni3WD::wheelRightSetSpeedMMPS**() and **Omni3WD::wheelLeftSetSpeedMMPS**()

Parameters:	unsigned int speedMMPS=0 The speed for the car moves,initialize it.
See:	**Omni3WD::setCarstat**() **Omni3WD::wheelBackSetSpeedMMPS**() **Omni3WD::wheelRightSetSpeedMMPS**() **Omni3WD::wheelLeftSetSpeedMMPS**()

unsigned int **setCarRotateLeft**(unsigned int speedMMPS=0)

Set the car for rotate left

This will lie within the range specified at **Omni3WD::setCarstat**() and **Omni3WD::setMotorAllBackoff**()

Parameters:	unsigned int speedMMPS=0 The speed for the car moves,initialize it.
See:	**Omni3WD::setCarstat**() **Omni3WD::setMotorAllBackoff**()

unsigned int **setCarRotateRight**(unsigned int speedMMPS=0)

Set the car for rotate right

This will lie within the range specified at **Omni3WD::setCarstat**() and

Omni3WD::setMotorAllAdvance()

Parameters:	unsigned int speedMMPS=0 The speed for the car moves,initialize it.
See:	**Omni3WD::setCarstat**() **Omni3WD::setMotorAllAdvance**()

unsigned int **getCarSpeedMMPS**() const

Get the car's speed

return:	The car's speed

unsigned int **setCarSpeedMMPS(**unsigned int speedMMPS=0,unsigned int ms=1000)

The car's speed be set

This will lie within the range specified at **Omni3WD::getCarSpeedMMPS**()

Parameters:	unsigned int speedMMPS=0 The speed for the car moves,initialize it. unsigned int ms=1000 The time the to moves the car at this speed
See:	**Omni3WD::getCarSpeedMMPS**()

unsigned int **setCarSlow2Stop**(unsigned int ms=1000)

Set the car stop in 1000 milliseconds

This will lie within the range specified at **Omni3WD::setCarSpeedMMPS**()

Parameters:	unsigned int ms=1000 The time to stop the car,initialize it
See:	**Omni3WD::getCarSpeedMMPS**()

unsigned int **wheelBackSetSpeedMMPS**(unsigned int speedMMPS=0,bool dir=DIR_ADVANCE)

Set the speed for the back wheel

This will lie within the range specified at **MotorWheel::setSpeedMMPS**()

Parameters:	unsigned int speedMMPS=0 The speed for the wheel to run,initialize it bool dir=DIR_ADVANCE The direction for the wheel to run
See:	**MotorWheel::setSpeedMMPS**()

unsigned int **wheelBackGetSpeedMMPS**() const

Get the speed of the back wheel

This will lie within the range specified at **MotorWheel::getSpeedMMPS**()

See:	**MotorWheel::getSpeedMMPS**()

unsigned int **wheelRightSetSpeedMMPS**(unsigned int speedMMPS=0,bool dir=DIR_ADVANCE)

Set the speed for the right wheel

This will lie within the range specified at **MotorWheel::setSpeedMMPS**()

Parameters:	unsigned int speedMMPS=0 The speed for the wheel to run,initialize it bool dir=DIR_ADVANCE The direction for the wheel to run
See:	**MotorWheel::setSpeedMMPS**()

unsigned int **wheelRightGetSpeedMMPS**() const

Get the speed of the right wheel

This will lie within the range specified at **MotorWheel::getSpeedMMPS**()

See:	**MotorWheel::getSpeedMMPS**()

unsigned int **wheelLeftSetSpeedMMPS**(unsigned int speedMMPS=0,bool dir=DIR_ADVANCE);

Set the speed for the left wheel

This will lie within the range specified at **MotorWheel::setSpeedMMPS**()

Parameters:	unsigned int speedMMPS=0 The speed for the wheel to run,initialize it bool dir=DIR_ADVANCE The direction for the wheel to run
See:	**MotorWheel::setSpeedMMPS**()

unsigned int **wheelLeftGetSpeedMMPS**() const

Get the speed of the left wheel

This will lie within the range specified at **MotorWheel::getSpeedMMPS**()

See:	**MotorWheel::getSpeedMMPS**()

bool **PIDEnable**(float kc=KC,float taui=TAUI,float taud=TAUD,unsigned int interval=1000)

Call the PID,make it work for the car

This will lie within the range specified at **MotorWheel::PIDEnable**()

Parameters:	Float kc Proportional term,initialize it Float taui Integral term Float taud Derivative term Unsigded int interval The time the PID work last
see :	**MotorWheel::PIDEnable**()

bool **PIDRegulate**()

Regulate the PID ,in order to adjust the speed of the Motor.

This will lie within the range specified at **MotorWheel:: PIDRegulate**()

see :	**MotorWheel:: PIDRegulate**()

void **delayMS**(unsigned int ms=100, bool debug=false)

The time used for the car work as the same action

In the function,every 10 milliseconds,it will call the function PIDRegulate once time

Parameters:	unsigned long ms=100 The time the action last bool debug=false A flag

void **debugger**(bool wheelBackDebug=true,bool wheelRightDebug=true,bool wheelLeftDebug=true) const

Debug the speed of the wheel

Car _state enum

Used to configure the behavior of a car.

Note that not all car can be configured in every state.

Variables:

STAT_UNKNOWN

The state of the car unknown

STAT_STOP

The car's state is stop

STAT_ADVANCE

The car's state is moves forward

STAT_BACKOFF

The car's state is get backoff

STAT_ROTATELEFT

The car's state is moves rotate left

STAT_ROTATERIGHT

The car's state is moves rotate right

STAT_RIGHT

The car's state is turn right

STAT_LEFT

The car's state is turn left

unsigned char **getCarStat()** const

Get the state of the car

return : The car's state

Motor _state enum

Used to configure the behavior of a motor.

Note that not all motors can be configured in every state.

Variables:

MOTORS_BRL

The switchmotorstat is back-right-left

MOTORS_LBR

The switchmotorstat is left-back-right

MOTORS_RLB

The switchmotorstat is right-left-back

unsigned char **getSwitchMotorsStat()** const

Get the state of the Motor

return : The motor's state

➢ Private parameters

MotorWheel***** _**wheelBack**

A point named wheelBack as the object of MotorWheel

MotorWheel***** _**wheelRight**

A point named wheelright as the object of MotorWheel

MotorWheel***** _**wheelLeft**

A point named wheelLeft as the object of MotorWheel

unsigned char _**carStat**

To save the car's state

unsigned char **setCarStat**(unsigned char stat)

Set the state of the car

Parameters:	unsigned char stat The state want to set
return :	Carstate if the stat in the range of the want STAT_UNKNOWN otherwise

unsigned char _**switchMotorsStat**

Switch the motors' state

unsigned char **setSwitchMotorsStat**(unsigned char switchMotorsStat)

Set the Motors' state

This will lie within the range specified at **Omni3WD::getSwitchMotorsStat**()

Parameters:	unsigned char switchMotorsStat The state want to set
See:	**Omni3WD::getSwitchMotorsStat**()

Omni3WD()

Construct a new R2DW instance.

void **demoActions**(unsigned int speedMMPS=100,unsigned int duration=5000,

unsigned int uptime=500,bool debug=false)

A demo function for three wheels car to show

```
void Omni3WD::demoActions(unsigned int speedMMPS,unsigned int duration,unsigned int uptime,bool
debug) {
    unsigned int (Omni3WD::*carAction[])(unsigned int speedMMPS)={
        &Omni3WD::setCarAdvance,    //set car moves forward
        &Omni3WD::setCarBackoff,    //set car moves Reverse
        &Omni3WD::setCarLeft,       //set car turn left
        &Omni3WD::setCarRight,      //set car turn right
        &Omni3WD::setCarRotateLeft,//set car rotate left
        &Omni3WD::setCarRotateRight //set car rotate right
    };
    for(int i=0;i<6;++i) {                  //there are six base actions
        (this->*carAction[i])(0);           //choose one of the six actions
        setCarSpeedMMPS(speedMMPS,uptime);          // set the speed for the car in this action
        delayMS(duration,debug);                    // The time used for the car moves at this
speed in this action
        setCarSlow2Stop(uptime);                        //set the car stop slowly in uptime
    }
    setCarStop();                                   //set car stop
    delayMS(duration,debug);                        //delay(duration) every 10 milliseconds
call the PIDRegulate once time
    switchMotorsLeft();                             //rotate left to change the wheel to work
}
```

➢ Omni3WD_test

Here's an example ,we use it to test a car with three wheels.after this ,you will More thorough understanding of the library

Simple code:

```
#include <MotorWheel.h>
#include <Omni3WD.h>
#include <Omni4WD.h>
#include <PID_Beta6.h>
#include <PinChangeInt.h>
#include <PinChangeIntConfig.h>              // Include the header files
/*
      Wheel3    //             \\ Wheel2



                          ==
                        Wheel1
 */
irqISR(irq1,isr1);                    // Intterrupt function.on the basis of the pulse    ,work for wheel1
MotorWheel wheel1(9,8,6,7,&irq1);             //This will create a MotorWheel object called Wheel1
                                                //Motor PWM:Pin9, DIR:Pin8, Encoder A:Pin6, B:Pin7
irqISR(irq2,isr2);
MotorWheel wheel2(10,11,12,13,&irq2);


irqISR(irq3,isr3);
MotorWheel wheel3(3,2,4,5,&irq3);
//MotorWheel wheel3(5,4,2,3,&irq3);
// why not this??   Because the pin 5,pin 6 control by timer 0


Omni3WD Omni(&wheel1,&wheel2,&wheel3);
                                              // This will create a Omni3WD object called Omni3WD.
                                              //You can then use any of its methods; for instance,
```

```
                                        // to control a Omni3WD attached to pins, you could write
void setup() {
    TCCR1B=TCCR1B&0xf8|0x01;    // Timer1.Pin9,Pin10 PWM 31250Hz
    TCCR2B=TCCR2B&0xf8|0x01;    // Timer2 .Pin3,Pin11 PWM 31250Hz
    Omni.PIDEnable(0.26,0.02,0,10);  // Enable PID
}
void loop() {
    Omni.demoActions(100,5000,1000,false);
                    //Call the demoActions speedMMPS=100    duration=5000    uptime =1000.
/*
    Omni.setCarLeft(0);
    Omni.setCarSpeedMMPS(300,1000);
    Omni.delayMS(10000,true);
    Omni.setCarSlow2Stop(1000);
    Omni.setCarRight(0);
    Omni.setCarSpeedMMPS(100,1000);
    Omni.delayMS(10000,true);
    Omni.setCarSlow2Stop(1000);

    Omni.setCarLeft(100);
    for(int i=0;i<1000;++i) {
        Omni.PIDRegulate();
        delay(10);
    }
    Omni.setCarRight(100);
    for(int i=0;i<1000;++i) {
        Omni.PIDRegulate();
        delay(10);
    }
   */
}
```

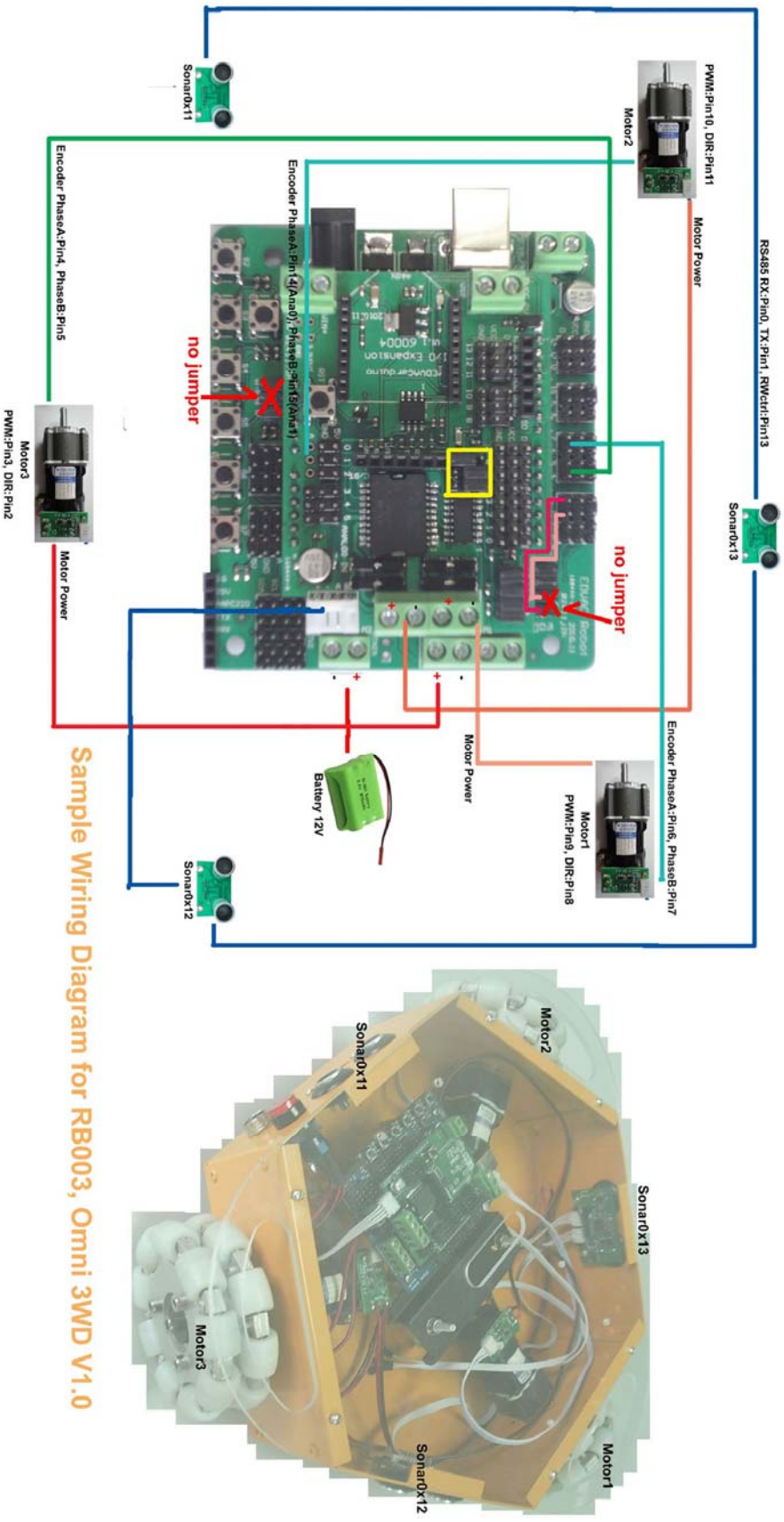


Diagram for Omni3WD_V1.0

➢ Omni3WD platform with 3 SONARS

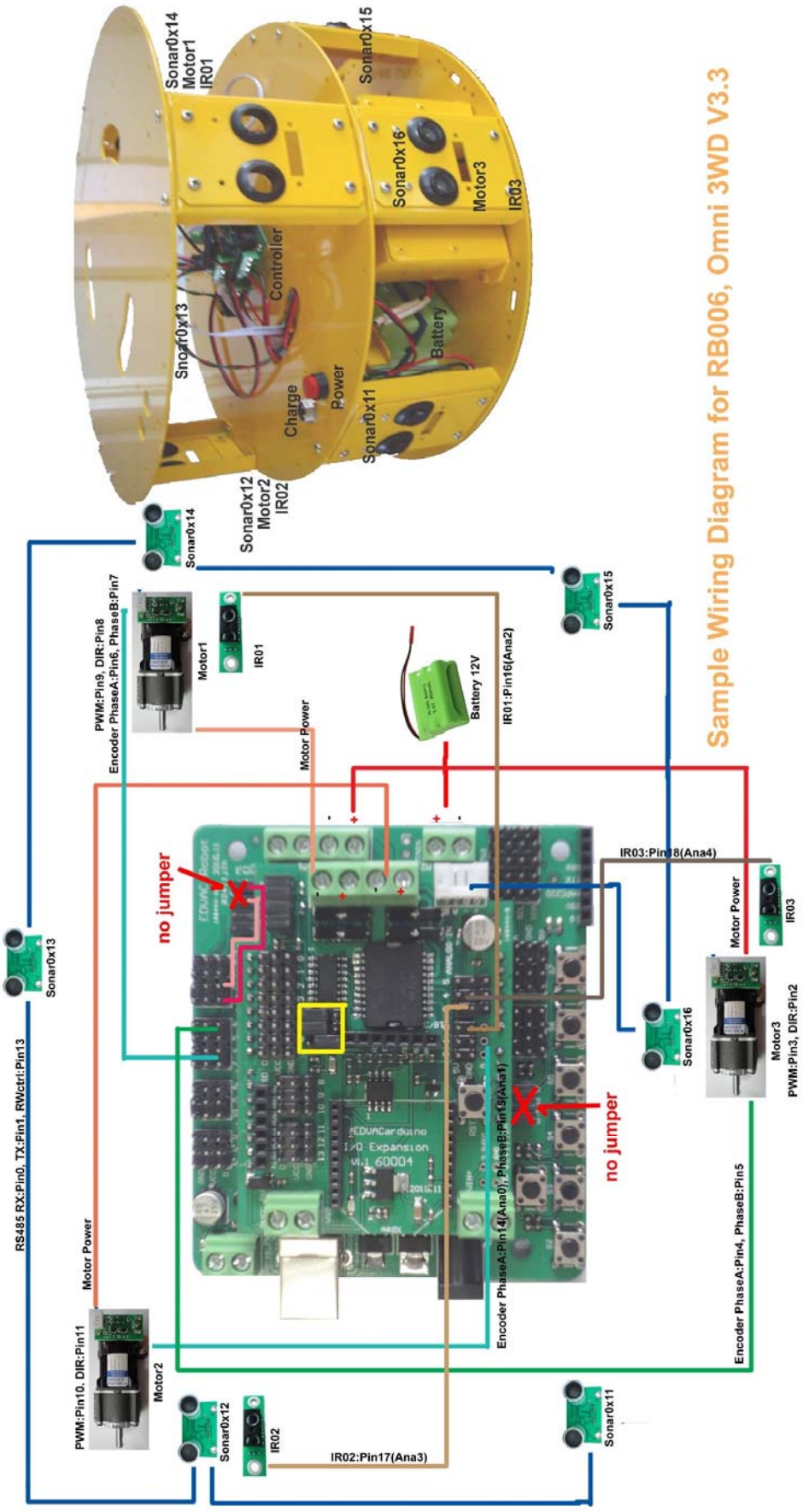
Look the above figure of simple Wiring Diagram for Omni3WD_V1.0. Tis code is matched for it

Diagram_Omni3WD_V1.0 code

```
/*********************************************************************/
/*
                                Power Switch

                                Sonar0x11
                              -------------------------
                             /                         \
                            /                           \
                           /                             \
                    M3    /                               \ M2
                    INT0 /                                 \INT1
                        /                                   \
                       /                                     \
                      /                                       \
                      \                                       /
                       \                                     /
                        \                                   /
                         \                                 /
                          \                               /
          Sonar0x12        \                             / Sonar0x13
                            \                           /
                             \                         /
                              -------------------------
                                       M1


 */
#include <fuzzy_table.h>
#include <PID_Beta6.h>
#include <PinChangeIntConfig.h>
#include <MotorWheel.h>
#include <Omni3WD.h>
#include <SONAR.h>          // Include the header files

/*******************************************/
// SONAR
```

```
SONAR sonar11(0x11),sonar12(0x12),sonar13(0x13); // Software initialization
                                                  //SONAR is be defined in <SONAR.h> as a class
unsigned short distBuf[3];                        // Used to save the data of the 3 sonars return;
void sonarsUpdate() {                             //the function to
    static unsigned char sonarCurr=1;    // A variable save a data used to flag the current of sonar
    if(sonarCurr==3) sonarCurr=1;
    else ++sonarCurr;
    if(sonarCurr==1) {              // The conditions is ture?
        distBuf[1]=sonar12.getDist();       // Get the value of distance from sonar12
        sonar12.trigger();                  // Trigger sonar12
    } else if(sonarCurr==2) {
        distBuf[2]=sonar13.getDist();       // Ger the value of distance from sonar13
        sonar13.trigger();                  // Trigger sonar13
    } else {
        distBuf[0]=sonar11.getDist();       // Ger the value of distance from sonar11
        sonar11.trigger();                  // Trigger sonar11
    }
}

/*********************************************/

/*********************************************/
// Motors

irqISR(irq1,isr1);
MotorWheel wheel1(9,8,6,7,&irq1);             // Pin9:PWM, Pin8:DIR, Pin6:PhaseA, Pin7:PhaseB

irqISR(irq2,isr2);
MotorWheel wheel2(10,11,14,15,&irq2);       // Pin10:PWM, Pin11:DIR, Pin14:PhaseA, Pin15:PhaseB

irqISR(irq3,isr3);
MotorWheel wheel3(3,2,4,5,&irq3);             // Pin3:PWM, Pin2:DIR, Pin4:PhaseA, Pin5:PhaseB
Omni3WD Omni(&wheel1,&wheel2,&wheel3);
                                    // This will create a Omni3WD object called Omni. then You
                                   // can use any of its methods; for instance, to
                                  // control a Omni3WD attached to pins, you could write
/*****************************************/
```

```
 /*****************************************/
// demo
unsigned long currMillis=0;
void demoWithSensors(unsigned int speedMMPS,unsigned int distance) {
    if(millis()-currMillis>SONAR::duration) {          // every 60ms call sonarUpdate once
        currMillis=millis();
        sonarsUpdate();
    }

    if(distBuf[1]<distance) {                          // If the left side have something
        if(Omni.getCarStat()!=Omni3WD::STAT_RIGHT) Omni.setCarSlow2Stop(500);
            Omni.setCarRight(speedMMPS);       // Set car turn right
    } else if(distBuf[2]<distance) {                   // If the right have something
        if(Omni.getCarStat()!=Omni3WD::STAT_LEFT) Omni.setCarSlow2Stop(500);
            Omni.setCarLeft(speedMMPS);        // Set car turn left
    } else if(distBuf[0]<distance) {                   // If the front have something
        if(Omni.getCarStat()!=Omni3WD::STAT_ROTATERIGHT) Omni.setCarSlow2Stop(500);
            Omni.setCarRotateRight(speedMMPS); // Set car rotateright
    } else {                                           // There is nothing around the car
        if(Omni.getCarStat()!=Omni3WD::STAT_ADVANCE) Omni.setCarSlow2Stop(500);
            Omni.setCarAdvance(speedMMPS);       // Set car moves advance
    }
        Omni.PIDRegulate();                            //PID regulate
}
/*****************************************/
// setup()
void setup() {
    TCCR1B=TCCR1B&0xf8|0x01;       // Pin9,Pin10 PWM 31250Hz
    TCCR2B=TCCR2B&0xf8|0x01;       // Pin3,Pin11 PWM 31250Hz
        SONAR::init(13);                    // Initial sonars
        Omni.PIDEnable(0.26,0.02,0,10);             // Enable PID
    }

/*****************************************/
// loop()
void loop() {
    demoWithSensors(80,30);                    // call the demo actions
}
```



Diagram_Omni3WD_V3.3

➢ Omni3WD platform with 6 SONARS

Look the above figure of simple Wiring Diagram for Omni3WD_V3.3. Tis code is matched for it

Diagram_Omni3WD_V3.3 code

```
/*****************************************************************/
/*
                                Power Switch
                                Sonar0x11
                             ----------------------
                            /                      \
                           /                        \
          Sonar0x16       /                          \     Sonar0x12
            M3,IR03      /                            \    M2,IR02
                        /                              \
                       /                                \
                      /                                  \
                     /                                    \
                    /                                      \
                    \                                      /
                     \                                    /
                      \                                  /
                       \                                /
                        \                              /
          Sonar0x15      \                            / Sonar0x13
                          \                          /
                           \                        /
                            ----------------------
                                                Sonar0x14
                                                M1,IR01
     */
#include <fuzzy_table.h>
#include <PID_Beta6.h>
#include <PinChangeInt.h>
#include <PinChangeIntConfig.h>
#include <MotorWheel.h>
#include <Omni3WD.h>
#include <SONAR.h>                      // Include the header files

/****************************************/
// SONAR
```

```
SONAR sonar11(0x11),sonar12(0x12),sonar13(0x13),sonar14(0x14),sonar15(0x15),sonar16(0x16);
                                                    // Software initialization
                                                     //SONAR is be defined in <SONAR.h> as a class

unsigned short distBuf[6];                          // Used to save the data of the 6 sonars return;
void sonarsUpdate() {
    static unsigned char sonarCurr=1;       // A variable save a data used to flag the current of sonar
    if(sonarCurr==3) sonarCurr=1;
    else ++sonarCurr;
    if(sonarCurr==1) {                      // The conditions is ture?
        distBuf[1]=sonar12.getDist();           // Get the value of distance from sonar12
        distBuf[4]=sonar15.getDist();
        sonar11.trigger();                      // Trigger sonar11
        sonar14.trigger();
    } else if(sonarCurr==2) {
        distBuf[2]=sonar13.getDist();
        distBuf[5]=sonar16.getDist();
        sonar12.trigger();
        sonar15.trigger();
    } else {
        distBuf[0]=sonar11.getDist();
        distBuf[3]=sonar14.getDist();
        sonar13.trigger();
        sonar16.trigger();
    }
}

/*****************************************/

/*****************************************/
unsigned char IRpin[]={16,17,18};       // Pin16, Pin17, Pin18
bool IRs[3]={false,false,false};

void initIRs() {                            // Initial the Pin mode
    pinMode(IRpin[0],INPUT);
    pinMode(IRpin[1],INPUT);
    pinMode(IRpin[2],INPUT);
}
```

```
void checkIRs() {                    // Read the Anti-drop Sonars
    for(int i=0;i<3;++i) {
        IRs[i]=digitalRead(IRpin[i]);   // Save datas
        Serial.print(IRs[i]);           // display the datas
    }
    Serial.println("");
}
/*********************************************/

/*********************************************/
// Wheels

irqISR(irq1,isr1);
MotorWheel wheel1(9,8,6,7,&irq1);            // Pin9:PWM, Pin8:DIR, Pin6:PhaseA, Pin7:PhaseB

irqISR(irq2,isr2);
MotorWheel wheel2(10,11,14,15,&irq2);       // Pin10:PWM, Pin11:DIR, Pin14:PhaseA, Pin15:PhaseB

irqISR(irq3,isr3);
MotorWheel wheel3(3,2,4,5,&irq3);            // Pin3:PWM, Pin2:DIR, Pin4:PhaseA, Pin5:PhaseB
//MotorWheel wheel3(5,4,2,3,&irq3);

Omni3WD Omni(&wheel1,&wheel2,&wheel3);
                                    // This will create a Omni3WD object called Omni. then You
                                   // can use any of its methods; for instance, to
                                  // control a Omni3WD attached to pins, you could write
/*****************************************/

/*****************************************/
// demo
unsigned long currMillis=0;
void demoWithSensors(unsigned int speedMMPS,unsigned int distance,unsigned int ms) {
    if(millis()-currMillis>SONAR::duration) {        //   Every 60 ms call the SonarsUpdate() once time
        currMillis=millis();
        sonarsUpdate();
    }
    checkIRs();                                    // check the Anti-drop sonars
```

```
    if(IRs[1] || IRs[2]) {                                  // The    Anti-drop return a High Value
        Omni.setCarBackoff(speedMMPS);          // get car back
        Omni.delayMS(ms);                 // delay “ms” ,every 10 ms call the PIDregulate() once time.
        if(IRs[1]) {
            Omni.setCarRotateLeft(speedMMPS);
        } else Omni.setCarRotateRight(speedMMPS);
        Omni.delayMS(ms);
    } else if(distBuf[1]<distance || distBuf[2]<distance) {    // the right side have something
         Omni.setCarLeft(speedMMPS);
     } else if(distBuf[4]<distance || distBuf[5]<distance) {    // Left side have something
         Omni.setCarRight(speedMMPS);
     } else if(distBuf[0]<distance || distBuf[3]<distance || IRs[0]) {
         Omni.setCarRotateRight(speedMMPS);
     } else {
         Omni.setCarAdvance(speedMMPS);
     }

     Omni.PIDRegulate();
     if(millis()%100==0) Omni.debugger();
}

/*****************************************/
// setup()
void setup() {
     TCCR1B=TCCR1B&0xf8|0x01;       // Pin9,Pin10 PWM 31250Hz
     TCCR2B=TCCR2B&0xf8|0x01;       // Pin3,Pin11 PWM 31250Hz

     SONAR::init(13);                          // Pin13 as RW Control
     initIRs();

     Omni.PIDEnable(0.26,0.02,0,10);     //Enable PID
     }
/*****************************************/
// loop()
void loop() {
     demoWithSensors(80,20,300);                    //Call the demo function
}
```

➢ Omni4WD

This page describes how to control the built-in **Omni4WD**. It does not describe how the **Omni4WD** work on your board. For more information on that, Please refer to **Omni4WD Class Reference**.

RB011_4WD Mecanum wheel mobile kit

*4WD 100mm Mecanum wheel
* Includes ultrasonic sensors and optional IR
*Suspension structure to ensure roadholding of each single wheel
*DC motors with encoders
*Microcontroller and IO expansion board
*Programmable with c,c++
*Based on Arduino microcontroller

*4WD 100mm Mecanum wheel
*DC motors with encoders
*Microcontroller and IO expansion board
*Flexible base plate ensuring roadholding of each single wheel
*Programmable with c,c++
*Based on Arduino microcontroller

RB009_4WD Mecanum wheel Simple Base

RB008_4WD Omni wheel Simple Base

*4WD 100mm Omni wheel
*DC motors with encoders
*Microcontroller and IO expansion board
*Idea platform to learn and build your omni wheel robot
*Easy to assemble
* Capable of omni direction movement and rotating
*Programmable with c,c++
*Based on Arduino microcontroller

This is a 4 wheel drive,Mecanum wheel mobile platform vehicle is stable and can be made to move in any direction and turn by varying the direction and speed of each wheel. Moving all four wheels in the same direction causes forward/backward movement, running left/right sides in opposite directions causes rotation, and running front and rear in opposite directions causes sideways movement.Its special way its rear wheels mounted ensure roadholding of each wheel.

➢ Omni4WD Class Reference

This document describes a car with four Motors. On the Introduction, you will know how to use the Omni4WD library to control the Motors,then to control the car

#include<MotorWheel.h>

Include the header file **MotorWheel.h**

This section gives a full listing of the capabilities of **Omni4WD**

Class Omni4WD

Interface for visit of peripherals

➢ Public functions

Omni4WD::Omni4WD(MotorWheel* wheelUL,MotorWheel* wheelLL,

MotorWheel* wheelLR,MotorWheel* wheelUR):

_wheelUL(wheelUL),_wheelLL(wheelLL),

_wheelLR(wheelLR),_wheelUR(wheelUR) {

setSwitchMotorsStat(MOTORS_FB);

}

Construct a new Omni4WD instance.

in your sketch. This will create a Omni4WD object called Omni4WD. You can then use any of its methods; for instance, to control a Omni4WD attached to pins, you could write

Parameters:	**MotorWheel* wheelUL** A point named **wheelUL** as the object of MotorWheel **MotorWheel* wheelLL** A point named **wheelLL** as the object of MotorWheel **MotorWheel* wheelLR** A point named **wheelLR** as the object of MotorWhee **MotorWheel* wheelUR** A point named **wheelUR** as the object of MotorWhee

unsigned char **switchMotors**()

Switch Motors to control

This will lie within the range specified at **Omni4WD::getSwitchMotorsStat**()

See:	**Omni3WD::getSwitchMotorsStat**()

unsigned char **switchMotorsReset**()

Reset for switch motors to control

unsigned int **setMotorAll**(unsigned int speedMMPS=0,bool dir=DIR_ADVANCE)

Set all motors' speed and direction

This will lie within the range specified at **Omni4WD:: wheelULSetSpeedMMPS**() and **Omni4WD::wheelLLSetSpeedMMPS**() and **Omni4WD::wheelLRSetSpeedMMPS**() and **Omni4WD::wheelURSetSpeedMMPS**()

Parameters:	unsigned int speedMMPS=0 The speed for the motor run,initialize it. bool dir=DIR_ADVANCE The direction the motor run
See:	**Omni4WD:: wheelULSetSpeedMMPS**() **Omni4WD::wheelLLSetSpeedMMPS**() **Omni4WD::wheelLRSetSpeedMMPS**() **Omni4WD::wheelURSetSpeedMMPS**()

unsigned int **setMotorAllStop**()

Set all Motors stop

This will lie within the range specified at **Omni4WD::setMotorAll**()

See:	**Omni4WD::setMotorAll**()

unsigned int **setMotorAllAdvance**(unsigned int speedMMPS=0)

Set all motors run forward

This will lie within the range specified at **Omni4WD::setMotorAll**()

Parameters:	unsigned int speedMMPS=0 The speed for the motor run,initialize it.
See:	**Omni4WD::setMotorAll**()

unsigned int **setMotorAllBackoff**(unsigned int speedMMPS=0)

Set all motors run back off

This will lie within the range specified at **Omni4WD::setMotorAll**()

Parameters:	unsigned int speedMMPS=0 The speed for the motor run,initialize it.
See:	**Omni4WD::setMotorAll**()

unsigned int **setCarStop**()

Stop the car

This will lie within the range specified at **Omni4WD::setMotorAll**() and **Omni4WD::setCarstat**()

See:	**Omni4WD::setMotorAll**() **Omni4WD::setCarstat**()

unsigned int **setCar**(unsigned int speedMMPS=0)

Set the car moves forward

Because the car have Four wheels ,so the car moves forward ,the wheels will have different state each other.

This will lie within the range specified at **Omni4WD:: wheelULSetSpeedMMPS**() and **Omni4WD::wheelLLSetSpeedMMPS**() and **Omni4WD::wheelLRSetSpeedMMPS**() and **Omni4WD::wheelURSetSpeedMMPS**() and **Omni4WD::setCarstat**()

Parameters:	unsigned int speedMMPS=0 The speed for the motor run,initialize it.
See:	**Omni4WD:: wheelULSetSpeedMMPS**() **Omni4WD::wheelLLSetSpeedMMPS**() **Omni4WD::wheelLRSetSpeedMMPS**() **Omni4WD::wheelURSetSpeedMMPS**() **Omni4WD::setCarstat**()

unsigned int **setCarBackoff**(unsigned int speedMMPS=0)

Set the car moves forward

Because the car have Four wheels ,so the car moves forward ,the wheels will have different state.

This will lie within the range specified at **Omni4WD:: wheelULSetSpeedMMPS**() and **Omni4WD::wheelLLSetSpeedMMPS**() and **Omni4WD::wheelLRSetSpeedMMPS**() and **Omni4WD::wheelURSetSpeedMMPS**() and **Omni4WD::setCarstat**()

Parameters:	unsigned int speedMMPS=0 The speed for the motor run,initialize it.
See:	**Omni4WD:: wheelULSetSpeedMMPS**() **Omni4WD::wheelLLSetSpeedMMPS**() **Omni4WD::wheelLRSetSpeedMMPS**() **Omni4WD::wheelURSetSpeedMMPS**() **Omni4WD::setCarstat**()

unsigned int **setCarLeft**(unsigned int speedMMPS=0)

Set the car turn Left

This will lie within the range specified at **Omni4WD:: wheelULSetSpeedMMPS**() and **Omni4WD::wheelLLSetSpeedMMPS**() and **Omni4WD::wheelLRSetSpeedMMPS**() and **Omni4WD::wheelURSetSpeedMMPS**() and **Omni4WD::setCarstat**()

Parameters:	unsigned int speedMMPS=0 The speed for the motor run,initialize it.
See:	**Omni4WD:: wheelULSetSpeedMMPS**() **Omni4WD::wheelLLSetSpeedMMPS**() **Omni4WD::wheelLRSetSpeedMMPS**() **Omni4WD::wheelURSetSpeedMMPS**() **Omni4WD::setCarstat**()

unsigned int **setCarRight**(unsigned int speedMMPS=0)

Set the car turn right

This will lie within the range specified at **Omni4WD:: wheelULSetSpeedMMPS**() and **Omni4WD::wheelLLSetSpeedMMPS**() and **Omni4WD::wheelLRSetSpeedMMPS**() and **Omni4WD::wheelURSetSpeedMMPS**() and **Omni4WD::setCarstat**()

Parameters:	unsigned int speedMMPS=0 The speed for the motor run,initialize it.
See:	**Omni4WD::wheelULSetSpeedMMPS**() **Omni4WD::wheelLLSetSpeedMMPS**() **Omni4WD::wheelLRSetSpeedMMPS**() **Omni4WD::wheelURSetSpeedMMPS**()

	Omni4WD::setCarstat()

unsigned int **setCarRotateLeft**(unsigned int speedMMPS=0)

Set the car rotate left

This will lie within the range specified at **Omni4WD::setCarstat**() and **mni4WD::setMotorAllBackoff**()

Parameters:	unsigned int speedMMPS=0 The speed for the car moves,initialize it.
See:	**Omni4WD::setCarstat**() **Omni4WD::setMotorAllBackoff**()

unsigned int **setCarRotateRight**(unsigned int speedMMPS=0)

Set the car for rotate right

This will lie within the range specified at **Omni4WD::setCarstat**() and

Omni4WD::setMotorAllAdvance()

Parameters:	unsigned int speedMMPS=0 The speed for the car moves,initialize it.
See:	**Omni4WD::setCarstat**() **Omni4WD::setMotorAllAdvance**()

unsigned int **setCarUpperLeft**(unsigned int speedMMPS=0)

Set the car upper left

This will lie within the range specified at **Omni4WD:: wheelULSetSpeedMMPS**() and

Omni4WD::wheelLLSetSpeedMMPS() and **Omni4WD::wheelLRSetSpeedMMPS**() and

Omni4WD::wheelURSetSpeedMMPS() and **Omni4WD::setCarstat**()

Parameters:	unsigned int speedMMPS=0 The speed for the motor moves,initialize it.
See:	**Omni4WD::wheelULSetSpeedMMPS**() **Omni4WD::wheelLLSetSpeedMMPS**() **Omni4WD::wheelLRSetSpeedMMPS**() **Omni4WD::wheelURSetSpeedMMPS**() **Omni4WD::setCarstat**()

unsigned int **setCarLowerLeft**(unsigned int speedMMPS=0)

Set the car Lower left

This will lie within the range specified at **Omni4WD::wheelULSetSpeedMMPS**() and **Omni4WD::wheelLLSetSpeedMMPS**() and **Omni4WD::wheelLRSetSpeedMMPS**() and **Omni4WD::wheelURSetSpeedMMPS**() and **Omni4WD::setCarstat**()

Parameters:	unsigned int speedMMPS=0 The speed for the motor run,initialize it.
See:	**Omni4WD::wheelULSetSpeedMMPS**() **Omni4WD::wheelLLSetSpeedMMPS**() **Omni4WD::wheelLRSetSpeedMMPS**() **Omni4WD::wheelURSetSpeedMMPS**() **Omni4WD::setCarstat**()

unsigned int **setCarUpperRight**(unsigned int speedMMPS=0)

Set the car upper Right

This will lie within the range specified at **Omni4WD::wheelULSetSpeedMMPS**() and **Omni4WD::wheelLLSetSpeedMMPS**() and **Omni4WD::wheelLRSetSpeedMMPS**() and **Omni4WD::wheelURSetSpeedMMPS**() and **Omni4WD::setCarstat**()

Parameters:	unsigned int speedMMPS=0 The speed for the motor run,initialize it.
See:	**Omni4WD::wheelULSetSpeedMMPS**() **Omni4WD::wheelLLSetSpeedMMPS**() **Omni4WD::wheelLRSetSpeedMMPS**() **Omni4WD::wheelURSetSpeedMMPS**() **Omni4WD::setCarstat**()

unsigned int **setCarLowerRight**(unsigned int speedMMPS=0)

Set the car Lower right

This will lie within the range specified at **Omni4WD::wheelULSetSpeedMMPS**() and **Omni4WD::wheelLLSetSpeedMMPS**() and **Omni4WD::wheelLRSetSpeedMMPS**() and **Omni4WD::wheelURSetSpeedMMPS**() and **Omni4WD::setCarstat**()

Parameters:	unsigned int speedMMPS=0

	The speed for the motor run,initialize it.
See:	**Omni4WD::wheelULSetSpeedMMPS**() **Omni4WD::wheelLLSetSpeedMMPS**() **Omni4WD::wheelLRSetSpeedMMPS**() **Omni4WD::wheelURSetSpeedMMPS**() **Omni4WD::setCarstat**()

unsigned int **getCarSpeedMMPS**() const

Get the car's speed

return:	The car's speed

unsigned int **setCarSpeedMMPS**(unsigned int speedMMPS=0,unsigned int ms=1000)

Set the speed of the car

This will lie within the range specified at **Omni4WD::getCarSpeedMMPS**()

Parameters:	unsigned int speedMMPS=0 The speed for the car moves,initialize it. unsigned int ms=1000 The time used for the to moves the car at this speed
See:	**Omni4WD::getCarSpeedMMPS**()

unsigned int **setCarSlow2Stop**(unsigned int ms=1000)

Set the car stop in 1000 milliseconds

This will lie within the range specified at **Omni4WD::setCarSpeedMMPS**()

Parameters:	unsigned int ms=1000 The time used for stop the car,initialize it
See:	**Omni4WD::getCarSpeedMMPS**()

unsigned int **getCarSpeedMMPS**() const

Get the car's speed

return:	The car's speed

unsigned int **setCarSpeedMMPS(**unsigned int speedMMPS=0,unsigned int ms=1000)

The car's speed be set

This will lie within the range specified at **Omni4WD::getCarSpeedMMPS**()

Parameters:	unsigned int speedMMPS=0 The speed for the car moves,initialize it. unsigned int ms=1000 The time used for the car moves at this speed
See:	**Omni4WD::getCarSpeedMMPS**()

unsigned int **setCarSlow2Stop**(unsigned int ms=1000)

Stop the car in 1000 milliseconds

This will lie within the range specified at **Omni4WD::setCarSpeedMMPS**()

Parameters:	unsigned int ms=1000 The time to stop the car,initialize it
See:	**Omni4WD::getCarSpeedMMPS**()

unsigned int **wheelULGetSpeedMMPS**() const

Get the speed of the upper left wheel

This will lie within the range specified at **MotorWheel::getSpeedMMPS**()

See:	**MotorWheel::getSpeedMMPS**()

unsigned int **wheelULSetSpeedMMPS**(unsigned int speedMMPS=0,bool dir=DIR_ADVANCE)

Set the speed for the upper left wheel

This will lie within the range specified at **MotorWheel::setSpeedMMPS**()

Parameters:	unsigned int speedMMPS=0 The speed for the wheel run,initialize it bool dir=DIR_ADVANCE The direction for the wheel run
See:	**MotorWheel::setSpeedMMPS**()

unsigned int **wheelLLGetSpeedMMPS**() const

Get the speed of the Lower left wheel

This will lie within the range specified at **MotorWheel::getSpeedMMPS**()

See:	**MotorWheel::getSpeedMMPS**()

unsigned int **wheelLLSetSpeedMMPS**(unsigned int speedMMPS=0,bool dir=DIR_ADVANCE)

Set the speed for the lower left wheel

This will lie within the range specified at **MotorWheel::setSpeedMMPS**()

Parameters:	unsigned int speedMMPS=0 The speed for the wheel run,initialize it bool dir=DIR_ADVANCE The direction for the wheel run
See:	**MotorWheel::setSpeedMMPS**()

unsigned int **wheelURGetSpeedMMPS**() const

Get the speed of the upper right wheel

This will lie within the range specified at **MotorWheel::getSpeedMMPS**()

See:	**MotorWheel::getSpeedMMPS**()

unsigned int **wheelURSetSpeedMMPS**(unsigned int speedMMPS=0,bool dir=DIR_ADVANCE)

Set the speed for the upper right wheel

This will lie within the range specified at **MotorWheel::setSpeedMMPS**()

Parameters:	unsigned int speedMMPS=0 The speed for the wheel run,initialize it bool dir=DIR_ADVANCE The direction for the wheel run
See:	**MotorWheel::setSpeedMMPS**()

unsigned int **wheelLRGetSpeedMMPS**() const

Get the speed of the lower right wheel

This will lie within the range specified at **MotorWheel::getSpeedMMPS**()

See:	**MotorWheel::getSpeedMMPS**()

unsigned int **wheelLRSetSpeedMMPS**(unsigned int speedMMPS=0,bool dir=DIR_ADVANCE)

Set the speed for the lower right wheel

This will lie within the range specified at **MotorWheel::setSpeedMMPS**()

Parameters:	unsigned int speedMMPS=0

	The speed for the wheel run,initialize it bool dir=DIR_ADVANCE The direction for the wheel run
See:	**MotorWheel::setSpeedMMPS**()

bool **PIDEnable**(float kc=KC,float taui=TAUI,float taud=TAUD,unsigned int interval=1000)

Call the PID,make it work for the car.Then this will work for every wheels

This will lie within the range specified at **MotorWheel::PIDEnable**()

Parameters:	Float kc Proportional term,initialize it Float taui Integral term Float taud Derivative term Unsigded int interval The time the PID work last
see :	**MotorWheel::PIDEnable**()

bool **PIDRegulate**()

Regulate the PID ,in order to adjust the speed of the Motor.

This will lie within the range specified at **MotorWheel:: PIDRegulate**()

see :	**MotorWheel:: PIDRegulate**()

void **delayMS**(unsigned int ms=100, bool debug=false)

The time used for the car work as the same action

In the function,every 10 milliseconds,it will call the function PIDRegulate once time

Parameters:	unsigned long ms=100 The time the action last,initialize it bool debug=false A flag

void **debugger**(bool wheelBackDebug=true,bool wheelRightDebug=true,bool wheelLeftDebug=true) const

Debug the speed of the wheel

Car _state enum

Used to configure the behavior of a car.

Note that not all car can be configured in every state.

Variables:

STAT_UNKNOWN

The state of the car unknown

STAT_STOP

The car's state is stop

STAT_ADVANCE

The car's state is moves forward

STAT_BACKOFF

The car's state is get back off

STAT_RIGHT

The car's state is turn right

STAT_LEFT

The car's state is turn left

STAT_ROTATELEFT

The car's state is moves rotate left

STAT_ROTATERIGHT

The car's state is moves rotate right

STAT_UPPERLEFT

The car's state is run upper left

STAT_LOWERLEFT

The car's state is moves lower left

STAT_LOWERRIGHT

The car's state is moves lower right

STAT_UPPERRIGHT

The car's state is moves upper right

unsigned char **getCarStat()** const

Get the state of the car

return : The car's state

Motor _state enum

Used to configure the behavior of a motor.

Note that not all motors can be configured in every state.

Variables:

MOTORS_FB

The switchmotorstat is front back

MOTORS_BF

The switchmotorstat is back front

unsigned char **getSwitchMotorsStat**() const

Get the state of the Motor

return : The motor's state

➢ Private parameters

MotorWheel***** _**wheelUL**

A point named _**wheelUL** as the object of MotorWheel

MotorWheel***** _**wheelLL**

A point named _wheelLL as the object of MotorWheel

MotorWheel***** _**wheelLR**

A point named _wheelLR as the object of MotorWheel

MotorWheel***** _**wheelUR**

A point named _wheelUR as the object of MotorWheel

unsigned char _**carStat**

To save the car's state

unsigned char **setCarStat**(unsigned char stat)

Set the state of the car

Parameters: unsigned char stat

The state want to set

return :	Carstate if the stat in the range of the want STAT_UNKNOWN otherwise

unsigned char **_switchMotorsStat**

Switch the motors' state

unsigned char **setSwitchMotorsStat**(unsigned char switchMotorsStat)

Set the Motors' state

This will lie within the range specified at **Omni4WD::getSwitchMotorsStat**()

Parameters:	unsigned char switchMotorsStat The state want to set
See:	**Omni4WD::getSwitchMotorsStat**()

Omni4WD()

Construct a new R2DW instance.

void **demoActions**(unsigned int speedMMPS=100,unsigned int duration=5000,unsigned int uptime=500,bool debug=false);

A demo function for four wheels car to show

```
void Omni4WD::demoActions(unsigned int speedMMPS,unsigned int duration,
        unsigned int uptime,bool debug) {
    unsigned int (Omni4WD::*carAction[])(unsigned int speedMMPS)={
        &Omni4WD::setCarAdvance,        // Car advance
        &Omni4WD::setCarBackoff,            //Car back off
        &Omni4WD::setCarLeft,           //Car turn left
        &Omni4WD::setCarRight,          //Car turn right
        &Omni4WD::setCarUpperLeft,      //Car upper left
        &Omni4WD::setCarLowerRight,         //Car lower right
        &Omni4WD::setCarLowerLeft,      //Car lower left
        &Omni4WD::setCarUpperRight,         //Car upper right
        &Omni4WD::setCarRotateLeft,     // Car rotate left
        &Omni4WD::setCarRotateRight         //Car rotate right
```

```
    };
    for(int i=0;i<10;++i) {                          //the car have 10 demo actions
        (this->*carAction[i])(0);                    // default parameters not available in function
pointer
        setCarSpeedMMPS(speedMMPS,uptime);       //in the uptime , the car's speed accelerate
from 0 to speedMMPS
        delayMS(duration,debug);                 //the car's state last "duration" times
        setCarSlow2Stop(uptime);                     //stop the car slowly in uptime
    }
    setCarStop();                                    //stop the car
    delayMS(duration);                               //delay(duration)
    switchMotors();                                  //switch the motors.
}
```

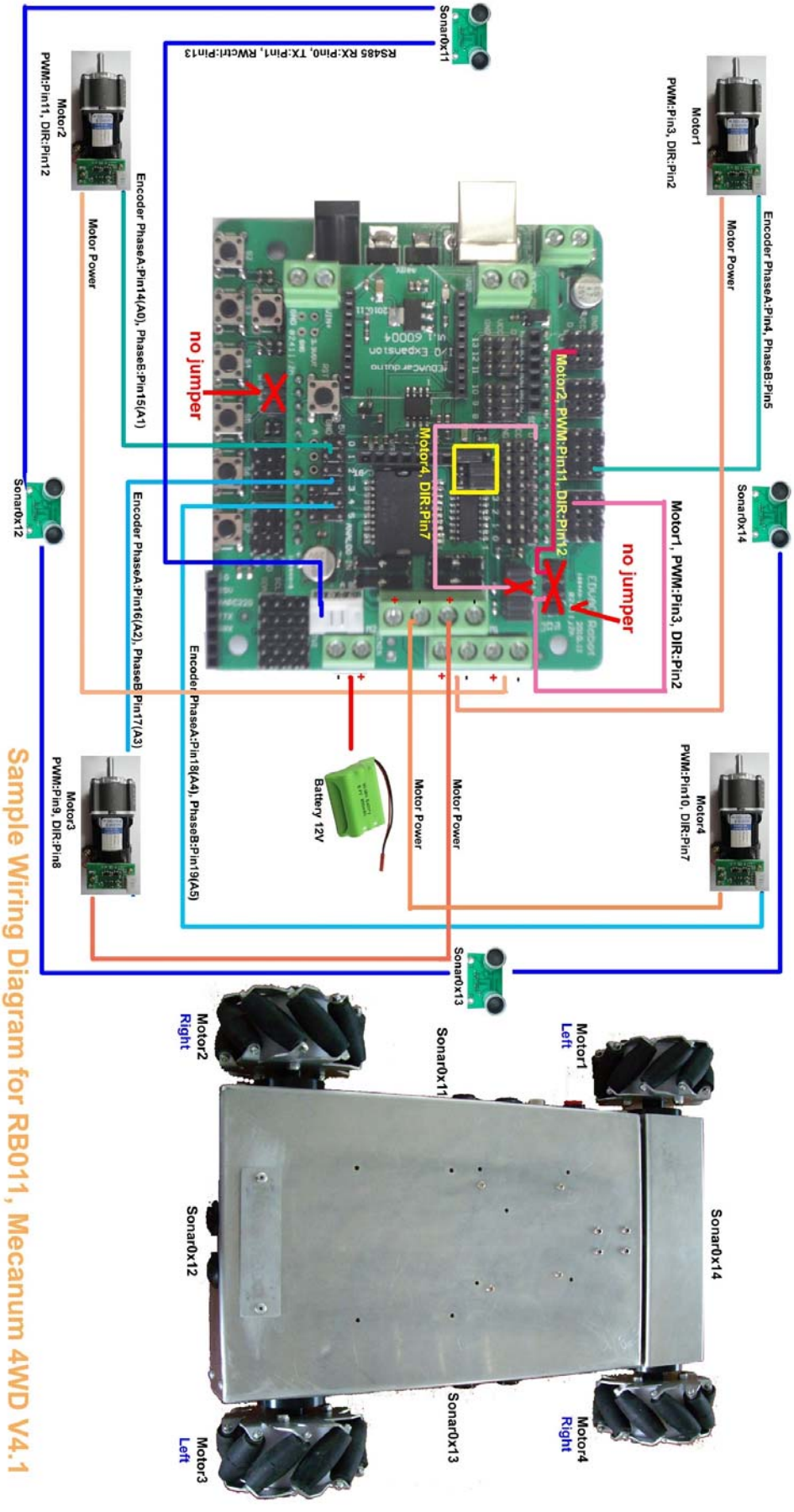
➢ Omni4WD_test

Here's an example ,we use it to test a car with four wheels.after this ,you will More thorough understanding of the library

Simple code:

```
#include <MotorWheel.h>
#include <Omni3WD.h>
#include <Omni4WD.h>
#include <PID_Beta6.h>
#include <PinChangeInt.h>
#include <PinChangeIntConfig.h>
                                        // Include the header files
/*
            \                                   /
    wheel1 \                                  /     wheel4
    Left        \                            /     Right









                /                             \
     wheel2 /                                   \     wheel3
     Right /                                     \     Left
 */
irqISR(irq1,isr1);                    // Intterrupt function.on the basis of the pulse,work for wheel1
MotorWheel wheel1(3,2,4,5,&irq1);
                                          //This will create a MotorWheel object called Wheel1
                                         //Motor PWM:Pin5, DIR:Pin4, Encoder A:Pin12, B:Pin13
irqISR(irq2,isr2);
MotorWheel wheel2(11,12,14,15,&irq2);


irqISR(irq3,isr3);
```

```
MotorWheel wheel3(9,8,16,17,&irq3);


irqISR(irq4,isr4);
MotorWheel wheel4(10,7,18,19,&irq4);


Omni4WD Omni(&wheel1,&wheel2,&wheel3,&wheel4);
                                                // This will create a Omni4WD object called Omni4WD.
                                                //You can then use any of its methods; for instance,
                                                 // to control a Omni4WD attached to pins, you could write
void setup() {
      //TCCR0B=TCCR0B&0xf8|0x01;       // warning!! it will change millis()
      TCCR1B=TCCR1B&0xf8|0x01;         // Pin9,Pin10 PWM 31250Hz
      TCCR2B=TCCR2B&0xf8|0x01;         // Pin3,Pin11 PWM 31250Hz
      Omni.PIDEnable(0.31,0.01,0,10);     // Enable PID
     }
void loop() {
      Omni.demoActions(200,5000,500,false);
                                          //Call the demoActions from the Class Omni4WD
                                           //speedMMPS=200     duration=5000
                                           //uptime =500         debug=false
     /*
      Omni.setCarLeft(0);
      Omni.setCarSpeedMMPS(200,500);
      Omni.delayMS(5000);
      Omni.setCarSlow2Stop(500);

      Omni.setCarRight(0);
      Omni.setCarSpeedMMPS(200,500);
      Omni.delayMS(5000);
      Omni.setCarSlow2Stop(500);
     */
     }
```



Sample Wiring Diagram for RB011 ,Mecanum 4WD V4.1

➢ 4WD platform with 4 SONAR

Look the above figure of simple Wiring Diagram for RB011 ,Mecanum 4WD V4.1. this code is matched for it

4WD platform with 4 SONAR code

```
#include <MotorWheel.h>
#include <Omni4WD.h>
#include <PID_Beta6.h>
#include <PinChangeInt.h>
#include <PinChangeIntConfig.h>
#include <SONAR.h>                    // Include the header files

/*
*********************************************************************************
                                        Sonar:0x12
                                ---------------------------------
                                |                               |
                          M3    |                               | M2
                                |                               |
                    Sonar:0x13  |                               | Sonar:0x11
                                |                               |
                                |                               | Power Switch
                                |                               |
                                |                               |
                                ---------------------------------
                                |                               |
                             M4 |                               | M1
                                |                               |
                                ---------------------------------
                                        Sonar:0x14
*********************************************************************************
*/
irqISR(irq1,isr1);       // Intterrupt function.on the basis of the pulse, work for wheel1
MotorWheel wheel1(3,2,4,5,&irq1);          //This will create a MotorWheel object called Wheel1
                                            //Motor PWM:Pin3, DIR:Pin2, Encoder A:Pin4, B:Pin5
irqISR(irq2,isr2);
MotorWheel wheel2(11,12,14,15,&irq2);

irqISR(irq3,isr3);
MotorWheel wheel3(9,8,16,17,&irq3);
```

```
irqISR(irq4,isr4);
MotorWheel wheel4(10,7,18,19,&irq4);


Omni4WD Omni(&wheel1,&wheel2,&wheel3,&wheel4);
            // This will create a Omni4WD object called Omni. then You can use any of its methods;
            // for instance, to control a Omni4WD attached to pins, you could write
SONAR sonar11(0x11),sonar12(0x12),sonar13(0x13),sonar14(0x14);     // Software initialization
                                                    //SONAR is be defined in <SONAR.h> as a class
unsigned short distBuf[4];                          // Used to save the data of the 3 sonars return

unsigned char sonarsUpdate() {
    static unsigned char sonarCurr = 1;       // A variable save a data used to flag the state of sonar
    if(sonarCurr==4) sonarCurr=1;
    else ++sonarCurr;
    if(sonarCurr==1) {                          // The conditions is ture?
        distBuf[1]=sonar12.getDist();           // Get the value of distance from sonar12
        sonar12.trigger();                      // Trigger sonar12
    } else if(sonarCurr==2) {
        distBuf[2]=sonar13.getDist();
        sonar13.trigger();
    } else if(sonarCurr==3){
        distBuf[3]=sonar14.getDist();
        sonar14.trigger();
    } else {
        distBuf[0]=sonar11.getDist();
        sonar11.trigger();
    }
        return sonarCurr;                           // Return the value
}

void goAhead(unsigned int speedMMPS){               // Car moves advance
    if(Omni.getCarStat()!=Omni4WD::STAT_ADVANCE) Omni.setCarSlow2Stop(300);
        Omni.setCarAdvance(0);                      // If the car's state is not advance.stop it
                                                    // else moves advance continue
        Omni.setCarSpeedMMPS(speedMMPS, 300);       // Set the car speed at 300
}

void turnLeft(unsigned int speedMMPS){
```

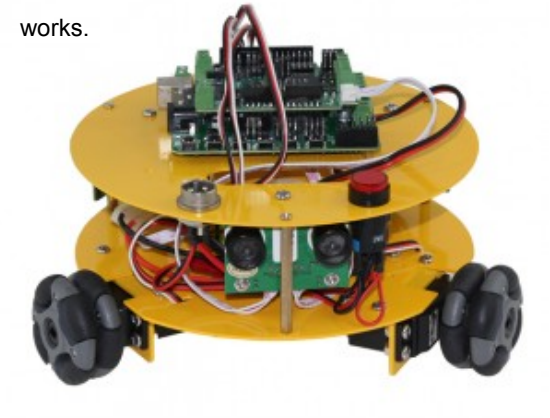
```
    if(Omni.getCarStat()!=Omni4WD::STAT_LEFT) Omni.setCarSlow2Stop(300);
        Omni.setCarLeft(0);
        Omni.setCarSpeedMMPS(speedMMPS, 300);
}

void turnRight(unsigned int speedMMPS){
    if(Omni.getCarStat()!=Omni4WD::STAT_RIGHT) Omni.setCarSlow2Stop(300);
        Omni.setCarRight(0);
        Omni.setCarSpeedMMPS(speedMMPS, 300);
}

void rotateRight(unsigned int speedMMPS){
    if(Omni.getCarStat()!=Omni4WD::STAT_ROTATERIGHT) Omni.setCarSlow2Stop(300);
        Omni.setCarRotateRight(0);
        Omni.setCarSpeedMMPS(speedMMPS, 300);
}

void rotateLeft(unsigned int speedMMPS){
    if(Omni.getCarStat()!=Omni4WD::STAT_ROTATELEFT) Omni.setCarSlow2Stop(300);
        Omni.setCarRotateLeft(0);
        Omni.setCarSpeedMMPS(speedMMPS, 300);
}

void allStop(unsigned int speedMMPS){
    if(Omni.getCarStat()!=Omni4WD::STAT_STOP) Omni.setCarSlow2Stop(300);
        Omni.setCarStop();
}

void backOff(unsigned int speedMMPS){

}

//void(*motion[8])(unsigned int speedMMPS) = {goAhead, turnLeft, rotateRight, rotateLeft,
    //turnRight, goAhead, rotateRight, backOff};

void(*motion[16])(unsigned int speedMMPS) = {goAhead, turnRight, goAhead, turnRight,
    turnLeft, goAhead, turnLeft, goAhead,
    rotateRight, rotateRight, turnRight, turnRight,
```

```
    rotateLeft, backOff, turnLeft, allStop};            // used the method of demotion

unsigned long currMillis=0;
void demoWithSensors(unsigned int speedMMPS,unsigned int distance) {
    unsigned char sonarcurrent = 0;
    if(millis()-currMillis>SONAR::duration + 20) {    // every 80 ms to call sonarUpdate once
        currMillis=millis();
        sonarcurrent = sonarsUpdate();
    }

    if(sonarcurrent == 4){
        unsigned char bitmap = (distBuf[0] < distance);   //right      Four of every byte
        bitmap |= (distBuf[1] < distance) << 1;   // back
        bitmap |= (distBuf[2] < distance) << 2;   // left
        bitmap |= (distBuf[3] < distance) << 3;   // front

        (*motion[bitmap])(speedMMPS);
    }
        Omni.PIDRegulate();               //PID regulate
}

void setup() {
    delay(2000);
    TCCR1B=TCCR1B&0xf8|0x01;      // Pin9,Pin10 PWM 31250Hz
    TCCR2B=TCCR2B&0xf8|0x01;      // Pin3,Pin11 PWM 31250Hz

    SONAR::init(13);
    //Omni.switchMotors();
    Omni.PIDEnable(2.0,1.0,0,10);    //PID enable
}

void loop() {
    //Omni.demoActions(250,5000,500,false);
    demoWithSensors(100,30);    //call the demo speed=300, distance=30.
}
```

➢ Servo Motor

This document describes a car with three servo Motors. On the Introduction, you will know how to control the servo Motors,then to control the car

Before you read this code,you should know about the Servo Motor Theory.To understand how the motor works.



*3WD 48mm Omni wheel
*Aluminum alloy fram
*Capable of rotation
*Includes Ultrasonic sensors
*Microcontroller and IO expansion board
*Programmable with C, C++
*Based on Arduino microcontroller

RB014_48mm 3WD Omni Wheel mobile robot kit

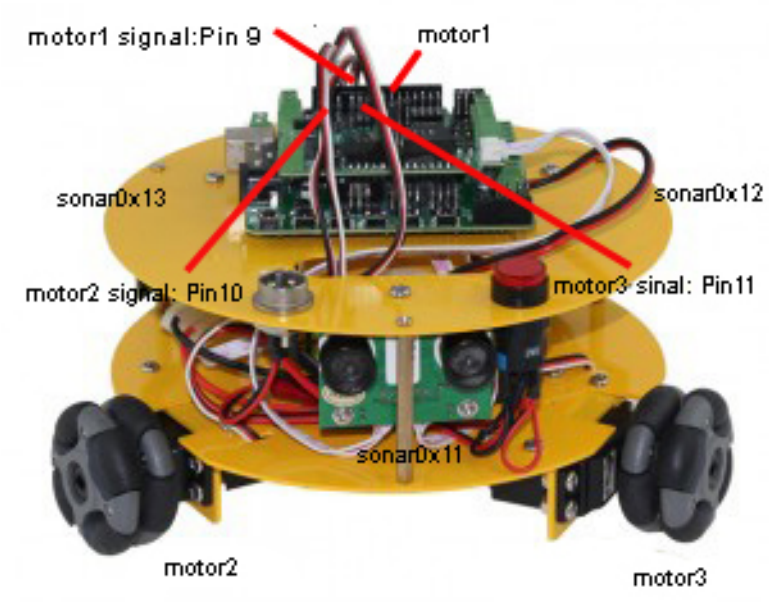
The 3WD 48mm Omni wheel mobile robot kit use three omni wheels with drive moving forward, backward, left, and right without change the direction and speed. Includes microcontroller and motors, it is programmable. Programming is performed by connecting to your PC and writting programs. There are still pre-drilled holes of screw and its firm aluminum alloy body makes it convenient and possible to add more levels.

➢ Simple code

```
#define   MOTOR1_E   9          //define the pin   9 as the motor1's pwm signal control pin
#define   MOTOR2_E 10           //define the pin 10 as the motor2's pwm signal control pin
#define   MOTOR3_E 11           //define the pin 11 as the motor3's pwm signal control pin
//*************************************************//
void goAhead(){
        analogWrite(MOTOR1_E, 48);    //stop run motor1
        analogWrite(MOTOR2_E, 62);    //forward run motor2
        analogWrite(MOTOR3_E, 34);    // Revese run motor3
}
//*************************************************//
void getBack(){
        analogWrite(MOTOR1_E,48); //stop tun motor1
        analogWrite(MOTOR2_E,32); // Reverse run motor2
        analogWrite(MOTOR3_E,64); //forward run motor3
}
//*************************************************//
void Left(){
        analogWrite(MOTOR1_E,36);    // Reverse run motor1
        analogWrite(MOTOR2_E,53);    //forward run motor2
```

```
        analogWrite(MOTOR3_E,53);    //forward run motor3
}
void Right(){
        analogWrite(MOTOR1_E,66);     //forward run motor1
        analogWrite(MOTOR2_E,40);     //Reverse run motor2
        analogWrite(MOTOR3_E,40);     //Reverse run motor3
}
//*************************************************//
void RotateRight(){
        analogWrite(MOTOR1_E,66);     //forward run motor1
        analogWrite(MOTOR2_E,62); //forward run motor2
        analogWrite(MOTOR3_E,64);         //forward run motor3
}
//*************************************************//
void RotateLeft(){
        analogWrite(MOTOR1_E,36); // reverse run motor 1
        analogWrite(MOTOR2_E,32); // reverse run motor 2
        analogWrite(MOTOR3_E,34);       // reverse run motor 3
}
//*************************************************//
void allStop(){
        analogWrite(MOTOR1_E, 48);     //stop run motor1
        analogWrite(MOTOR2_E, 44);       //stop run motor2
        analogWrite(MOTOR3_E, 46);       //stop run motor3
}
//*************************************************//
void (*motion[7])()={   goAhead,getBack,Left,Right,RotateLeft,RotateRight,allStop};
void demotion(){
        for(int i=0;i<7;i++){          // The demotion have 8 actions
             (*motion[i])();           // call the action
             delay(3000);              // Each action last 3000 milliseconds
        }
}
//*************************************************//
void setup() {
      TCCR1B=TCCR1B&0xf8|0x04;    //PIN 9 and PIN 10 cotroled by Timer1      122HZ
      TCCR2B=TCCR2B&0xf8|0x06;    //PIN 11 and PIN 3 cotroled by Timer2      122HZ
```

```
    pinMode(MOTOR1_E, OUTPUT);      // Define the pin Mode as OUTPUT
    pinMode(MOTOR2_E, OUTPUT);
    pinMode(MOTOR3_E, OUTPUT);
     Serial.begin(19200);
}
void loop(){
    demotion();     // Call the demotion
}
```



Sample Wiring Diagram for 3WD omni wheel mobile robot

➢ Servo_3WD platform with 3 SONAR

Look the above figure of simple Wiring Diagram for 3WD omni wheel mobile robot. Tis code is matched for it

Servo_3WD omni wheel mobile robot code

```
#include <SONAR.h>          // Include the header files

SONAR sonar11(0x11),sonar12(0x12),sonar13(0x13);        // Software initialization
                                                        //SONAR is be defined in <SONAR.h> as a class
#define   MOTOR1_E   9         //define the pin   9 as the motor1's pwm signal control pin
#define   MOTOR2_E 10          //define the pin 10 as the motor2's pwm signal control pin
#define   MOTOR3_E 11          //define the pin 11 as the motor3's pwm signal control pin
//************************************************//
unsigned short distBuf[3];
unsigned char sonarUpdate(){
            static unsigned int sonarCurr=1;
          if(sonarCurr==3) sonarCurr=1;
            else ++sonarCurr;
              Serial.println(sonarCurr);
             if(sonarCurr==1) {
                distBuf[0]=sonar11.getDist();   //Save the data of the distance get from Sonar11.
                sonar12.trigger();              // Tregger sonar12
          }else if(sonarCurr==2){
                distBuf[1]=sonar12.getDist();
                sonar13.trigger();              // trigger the sonar13
          }else if(sonarCurr==3){
                distBuf[2]=sonar13.getDist();   //according to the address to read the serial.
                sonar11.trigger();              // trigger the sonar11
            }
      return sonarCurr;
}
//************************************************//
void goAhead(){
         analogWrite(MOTOR1_E, 48);     //stop motor1
         analogWrite(MOTOR2_E, 62);     //forward motor2
         analogWrite(MOTOR3_E, 34);     // Revese motor3
}
//************************************************//
void getBack(){
```

```
        analogWrite(MOTOR1_E,48);
        analogWrite(MOTOR2_E,32);
        analogWrite(MOTOR3_E,64);
}
//*************************************************//
void turnLeft(){
        analogWrite(MOTOR1_E,36);   //Revese
        analogWrite(MOTOR2_E,53);   //forward
        analogWrite(MOTOR3_E,53);   //forward
}
void turnRight(){
        analogWrite(MOTOR1_E,66);    //forward
        analogWrite(MOTOR2_E,40);    //Revese
        analogWrite(MOTOR3_E,40);    //Revese
}
//*************************************************//
void RotateRight(){
        analogWrite(MOTOR1_E,66);   //forward
        analogWrite(MOTOR2_E,62);   //forward
        analogWrite(MOTOR3_E,64);   //forward
}
//*************************************************//
void RotateLeft(){
        analogWrite(MOTOR1_E,36);      //Revese
        analogWrite(MOTOR2_E,32);      //Revese
        analogWrite(MOTOR3_E,34);      //Revese
}
//*************************************************//
void judge(){
        if(distBuf[0]>=30){
                if(distBuf[1]<=10 && distBuf[2]>10) turnRight();
                else if(distBuf[2]<=10 && distBuf[1]>10) turnLeft();
                else if(distBuf[1]<=10 && distBuf[2]<=10) RotateLeft();
                else goAhead();
        }else RotateLeft();
}
//*************************************************//
void allStop(){
```

```
        analogWrite(MOTOR1_E, 48);     // stop the motor1
        analogWrite(MOTOR2_E, 44);     // stop the motor2
        analogWrite(MOTOR3_E, 46);     // stop the motor3
}
//*************************************************//
void (*motion[8])()={   goAhead,RotateLeft,turnRight,RotateLeft,turnLeft,RotateLeft,judge,allStop};
//change the

 void demowithSosars(){
     unsigned char sonarcurrent=0;
   if(millis()-currMillis>SONAR::duration){     //judge if the time more than SONAR::duration;
                  currMillis=millis();
                  sonarcurrent= sonarUpdate();              //if the requirement was ture call the function;
     }
     if(sonarcurrent==3){
       unsigned char bitmap = (distBuf[0] < 20);//front
       bitmap |= (distBuf[1]<20) <<1;                //left
       bitmap |= (distBuf[2]<20) <<2;                //right
       Serial.print("bitmap=");
       Serial.println(bitmap,DEC);
       (*motion[bitmap])();
     }
}
//*************************************************//
void setup() {
     TCCR1B=TCCR1B&0xf8|0x04;
     TCCR2B=TCCR2B&0xf8|0x06;
     pinMode(MOTOR1_E, OUTPUT);
     pinMode(MOTOR2_E, OUTPUT);
     pinMode(MOTOR3_E, OUTPUT);
     SONAR::init();          //call the init() from SONAR.h;
     delay(2000);
     Serial.begin(19200);
}
void loop(){
     demowithSosars();
     //delay(200);
}
```