make a figure

Robotics and Electronics

ManShow – RC1

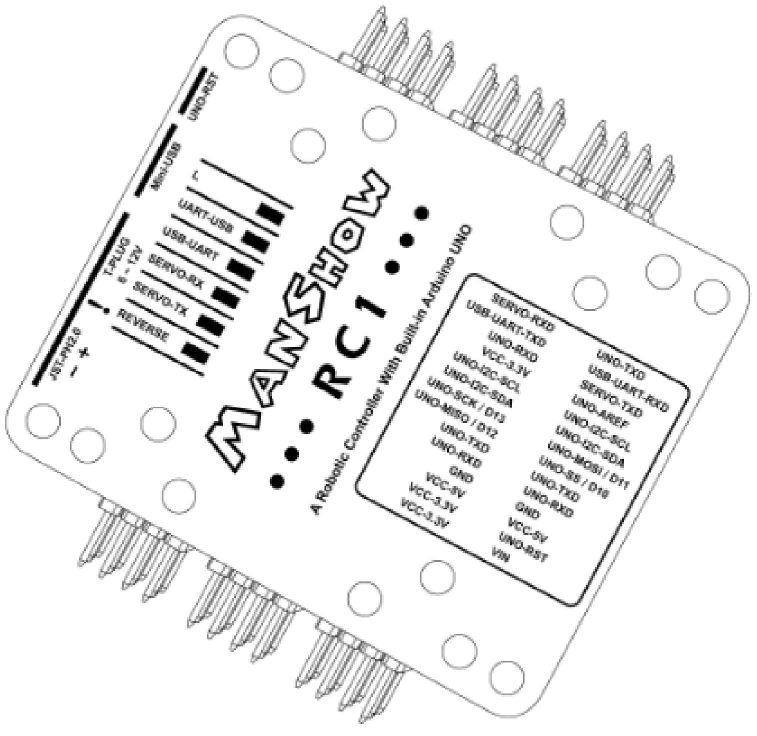
Introduction

ManShow-RC1 is mainly designed for humanoid robots, bionic robots and arms with a highly integrated structure(of course, it can also be used for other robotic systems), which integrates Arduino UNO R3 and **SoftServo** system(for 24 PWM-Style servos). The rich Arduino resources, standard connectors(for sensors) and servo control system(flexible, high-efficiency and stable), which make it easier and quicker to creat your robots.

Specification

Supply Voltage	: DC 6 ~ 12V (Mini-T Plug or JST-PH2.0-2P)
Size	: 58.2mm * 62.2mm * 23.5mm (L*W*H)

(Note: The size and mounting holes of circuit board are designed in strict compliance with the rule of **Gicren Robotics and Electronics**. **ManShow-RC1** is not a standard device of **Gicren**'s product line, so it does not fully comply with the rule.）



Internal Communications

There are two communication interfaces between Arduino UNO R3 and **SoftServo** system(for 24 PWM-Style servos): I2C(fixed, 100KHz) and UART(selectable, 1200 ~ 115200bps). These two communication interfaces share the same user registers, and the baud rate of the UART can be detected automatically. Please refer to the corresponding communication protocols for details(**GI2C_Vxx** / **G485_Vxx**).

Features

- - - For SoftServo - - -

- All the PWM signals output at the same time(in other words, they almost have the same rising edge), and the actual resolution is up to 1μs.
- The signal coupling is so low that it is difficult to detect visually, just like all the PWM signals are generated by separate hardware.
- It integrates the *Soft-Start* function. As we all know, the initial position of the *PWM-Style* servo is typically unknown. *Conventional-Start* mode makes servo rotate to the goal position with full speed, which is unsatisfactory in most robotic applications. *Soft-Start* mode can make servo rotate slowly when the current position is unknown, but need more current in a short time. It is suggested that all the servos should be started in time-sharing mode to reduce the electric current when you need to control several servos.
- The cycle, the minimum *PWM-Effective-Width* and the maximum *PWM-Effective-Width* can be set flexibly, which makes all the *PWM-Style* servos compatible.
- It is based on a subdivision algorithm, the maximum subdivision value is up to 250. The position and the speed of each channel can be controlled separately, which is conductive to robot dynamic control. The speed depends on the subdivision(see below).
- All the PWM signals can be terminated by setting the equivalent position value to 251 at any time, which is extremely useful for protecting the servo.
- It integrates a stable and high-efficiency UART, and the baud rate can be detected automatically. There is no instruction packet loss when the baud rate is not higher than 28800bps, otherwise the first instruction packet will be discarded. Furthermore, it is recommended to send the sync bytes(0xff 0xff 0xff 0xff) first when the baud rate is higher than 28800bps.
- It integrates a stable and high-efficiency I2C.

- - - For Arduino UNO R3 - - -

- Most of the digital and analog ports are extended in JST-PH2.0-3P(Digital Port: 10 channels / Analog Port: 4 channels), which make it easier to connect sensors(pins are defined as 1-SIGNAL, 2-VCC and 3-GND, and marked on the silkscreen).
- The power supply of JST-PH2.0-3P can be selected flexiblely(Digital Ports: 5V or Vin / Analog Ports: 3.3V or 5V)。

- The remaining pins of Arduino UNO R3 for extended-applications are led out on header(2*14 pins, 2.54mm pitch, simplified as "Expansion-Port" below), which can extend three communication interfaces(UART, I2C and SPI), establish three types of connections (USB<->UART<->UNO, USB<->UART<->SoftServo or UNO<->UART<->SoftServo), set the reference voltage for the ADC of Arduino UNO R3 to 3.3V and so on. For details about Arduino UNO R3, please visit: www.arduino.cc.

User Register Definition(For SoftServo)

Address	Name	Store	Access	Initial Value	Description
1(0x01)	ProtocolVer	ROM	R	0x__	Protocol Version (UART / I2C)
2(0x02)	DeviceMainClass	ROM	R	0x01	Device Main Class (0x01 : Main Controller)
3(0x03)	DeviceSubClass	ROM	R	0x01	Device Sub Class (0x01 : ManShow-RC Series)
4(0x04)	HardwareVer	ROM	R	0x1_	Hardware Version (0x1_ : ManShow-RC1)
5(0x05)	SoftwareVer	ROM	R	0x1_	Software Version (0x1_ : ManShow-RC1)
6(0x06)	Command	RAM	W	0x00	Command Byte
7(0x07)	ID	ROM	R/W	0x01	Device Address (1~126, 0 : Broadcast ID)
8(0x08)	PWM_Cycle	ROM	R/W	0x14	The cycle of the PWM signal (5~250, x1ms)
9(0x09)	PWM_MinL	ROM	R/W	0xf4	The lower byte of the minimum *PWM-Effective-Width*(1)
10(0x0a)	PWM_MinH	ROM	R/W	0x01	The higher byte of the minimum *PWM-Effective-Width*
11(0x0b)	PWM_MaxL	ROM	R/W	0xc4	The lower byte of the maximum *PWM-Effective-Width*
12(0x0c)	PWM_MaxH	ROM	R/W	0x09	The higher byte of the maximum *PWM-Effective-Width*
13(0x0d)	Position1	RAM	R/W	0xfb	The equivalent position of Channel 1 (0 ~ 251)
14(0x0e)	Position2	RAM	R/W	0xfb	The equivalent position of Channel 2 (0 ~ 251)
15(0x0f)	Position3	RAM	R/W	0xfb	The equivalent position of Channel 3 (0 ~ 251)
16(0x10)	Position4	RAM	R/W	0xfb	The equivalent position of Channel 4 (0 ~ 251)
17(0x11)	Position5	RAM	R/W	0xfb	The equivalent position of Channel 5 (0 ~ 251)

18(0x12)	Position6	RAM	R/W	0xfb	The equivalent position of Channel 6 (0 ~ 251)
19(0x13)	Position7	RAM	R/W	0xfb	The equivalent position of Channel 7 (0 ~ 251)
20(0x14)	Position8	RAM	R/W	0xfb	The equivalent position of Channel 8 (0 ~ 251)
21(0x15)	Position9	RAM	R/W	0xfb	The equivalent position of Channel 9 (0 ~ 251)
22(0x16)	Position10	RAM	R/W	0xfb	The equivalent position of Channel 10 (0 ~ 251)
23(0x17)	Position11	RAM	R/W	0xfb	The equivalent position of Channel 11 (0 ~ 251)
24(0x18)	Position12	RAM	R/W	0xfb	The equivalent position of Channel 12 (0 ~ 251)
25(0x19)	Position13	RAM	R/W	0xfb	The equivalent position of Channel 13 (0 ~ 251)
26(0x1a)	Position14	RAM	R/W	0xfb	The equivalent position of Channel 14 (0 ~ 251)
27(0x1b)	Position15	RAM	R/W	0xfb	The equivalent position of Channel 15 (0 ~ 251)
28(0x1c)	Position16	RAM	R/W	0xfb	The equivalent position of Channel 16 (0 ~ 251)
29(0x1d)	Position17	RAM	R/W	0xfb	The equivalent position of Channel 17 (0 ~ 251)
30(0x1e)	Position18	RAM	R/W	0xfb	The equivalent position of Channel 18 (0 ~ 251)
31(0x1f)	Position19	RAM	R/W	0xfb	The equivalent position of Channel 19 (0 ~ 251)
32(0x20)	Position20	RAM	R/W	0xfb	The equivalent position of Channel 20 (0 ~ 251)
33(0x21)	Position21	RAM	R/W	0xfb	The equivalent position of Channel 21 (0 ~ 251)
34(0x22)	Position22	RAM	R/W	0xfb	The equivalent position of Channel 22 (0 ~ 251)
35(0x23)	Position23	RAM	R/W	0xfb	The equivalent position of Channel 23 (0 ~ 251)
36(0x24)	Position24	RAM	R/W	0xfb	The equivalent position of Channel 24 (0 ~ 251)
37(0x25)	Subdivision1	RAM	R/W	0x01	The equivalent speed of Channel 1 (1 ~ 250)
38(0x26)	Subdivision2	RAM	R/W	0x01	The equivalent speed of Channel 2 (1 ~ 250)
39(0x27)	Subdivision3	RAM	R/W	0x01	The equivalent speed of Channel 3 (1 ~ 250)
40(0x28)	Subdivision4	RAM	R/W	0x01	The equivalent speed of Channel 4 (1 ~ 250)
41(0x29)	Subdivision5	RAM	R/W	0x01	The equivalent speed of Channel 5 (1 ~ 250)
42(0x2a)	Subdivision6	RAM	R/W	0x01	The equivalent speed of Channel 6 (1 ~ 250)

43(0x2b)	Subdivision7	RAM	R/W	0x01	The equivalent speed of Channel 7 (1 ~ 250)
44(0x2c)	Subdivision8	RAM	R/W	0x01	The equivalent speed of Channel 8 (1 ~ 250)
45(0x2d)	Subdivision9	RAM	R/W	0x01	The equivalent speed of Channel 9 (1 ~ 250)
46(0x2e)	Subdivision10	RAM	R/W	0x01	The equivalent speed of Channel 10 (1 ~ 250)
47(0x2f)	Subdivision11	RAM	R/W	0x01	The equivalent speed of Channel 11 (1 ~ 250)
48(0x30)	Subdivision12	RAM	R/W	0x01	The equivalent speed of Channel 12 (1 ~ 250)
49(0x31)	Subdivision13	RAM	R/W	0x01	The equivalent speed of Channel 13 (1 ~ 250)
50(0x32)	Subdivision14	RAM	R/W	0x01	The equivalent speed of Channel 14 (1 ~ 250)
51(0x33)	Subdivision15	RAM	R/W	0x01	The equivalent speed of Channel 15 (1 ~ 250)
52(0x34)	Subdivision16	RAM	R/W	0x01	The equivalent speed of Channel 16 (1 ~ 250)
53(0x35)	Subdivision17	RAM	R/W	0x01	The equivalent speed of Channel 17 (1 ~ 250)
54(0x36)	Subdivision18	RAM	R/W	0x01	The equivalent speed of Channel 18 (1 ~ 250)
55(0x37)	Subdivision19	RAM	R/W	0x01	The equivalent speed of Channel 19 (1 ~ 250)
56(0x38)	Subdivision20	RAM	R/W	0x01	The equivalent speed of Channel 20 (1 ~ 250)
57(0x39)	Subdivision21	RAM	R/W	0x01	The equivalent speed of Channel 21 (1 ~ 250)
58(0x3a)	Subdivision22	RAM	R/W	0x01	The equivalent speed of Channel 22 (1 ~ 250)
59(0x3b)	Subdivision23	RAM	R/W	0x01	The equivalent speed of Channel 23 (1 ~ 250)
60(0x3c)	Subdivision24	RAM	R/W	0x01	The equivalent speed of Channel 24 (1 ~ 250)
61(0x3d)	CurrentVoltage	RAM	R	0x__	8 times the supply voltage

1: *PWM-Effective-Width* is the high-level width of the PWM signal, which is typically from 500μs to 2500μs.

Command Byte Definition

What is the role of the command byte plays? Firstly, 100 user registers may not be able to meet the needs of all functions. The command byte is reserved for unexpected needs. Theoretically, there are 255 types of operations can be extended by changing the command byte. Secondly, sometimes it is hoped that some special operations can be performed by changing a command byte with some special values(easier to remember). Note that the software version must be higher than V12 when you set the command byte to “0xf0” or “0xf1”.

- 0xbf : Enable the *Soft-Start* mode.
- 0xbc : Enable the *Conventional-Start* mode(the default start mode after power-on).
- 0xcc : Synchronize the clock which is responsible for generating PWM signal.
- 0xf0 : When the PWM signal output again after termination, the breakpoint of the equivalent position will be used as the reference point.
- 0xf1 : When the PWM signal output again after termination, the middle position will be used as the reference point.

Technical Q & A

- **How to evaluate a PWM-Style servo controller**?

1. The *PWM-Effective-Width* of servo is not always the same. Taking the compatibility into account, it is necessary that the range of the *PWM-Effective-Width* can be set flexibly.
2. For the standard servo, the cycle of the PWM signal is typically 20ms, but a higher frequency PWM signal can be received by the digital servo. When an application system requires a higher response rate, the cycle of the PWM signal needs to be shorten. That is to say, the cycle of the PWM signal should also be able to be set flexibly.
3. It is hoped that all the PWM signals can output at the same time and the signal coupling could not be detected visually.
4. In some dynamic application systems, especially for the humanoid robot, whose dynamic balance is achieved by the inertia force, it is hoped that the position and the speed of each channel can be separately controlled at any time.
5. Most of the servo controllers store the servo action code in the EEPROM or Flash ROM, perhaps, it is simpler for operation. In our opinion, the servo action code of a robotic application system is better to be dynamically generated combining the sensors and intelligent algorithms. In addition, some features of the communication also need to be evaluated, such as the simplicity, the real-time capability and the stability.
6. In order to protect the servo, the PWM signal must be able to be terminated at any time. It is often seen that servo is damaged by the impact force. Theoretically, if there is any abnormal impact force, the PWM signal must be terminated. For example, once the robot is in the falling state, which can be detected by the accelerometer sensor, if the PWM signal can be immediately terminated to make the servo working in the coast or brake state, the rigid impact force would be avoided.

- **What is the mapping between the PWM-Effective-Width, the equivalent position and the actual position?**

If the *PWM-Effective-Width* is directly used as a parameter of an instruction packet, each position needs two bytes to describe. The theoretical position-controlled resolution can be improved to the microsecond level, however, it would greatly increase the length of the instruction packet. Taking into account the trade-off between the position-controlled resolution and the real-time capability of the communication, what stated below seems to be a reasonable solution. In addition, the PWM signal can be terminated when the equivalent position value is set to 251, and it is strongly recommended to use the *PWM-Style* servo which is locked by the periodic PWM signal(not a single PWM signal).

The variation of the *PWM-Effective-Width* is always divided into 250 parts. For example, the *PWM-Effective-Width* is from 500μs to 2500μs, and the corresponding angle is from 0° to 180°. The mapping is as follows ("X" is from 0 to 250).

PWM-Effective-Width (μs)	Equivalent Position	Actual Position (°)
500	0	0
2500	250	180
(2500-500)*X/250+500	X	180*X/250
Low-Level	251	Coast, Brake or Lock (according to the servo)

- **What is the subdivision in the servo controller**

The subdivision is the equivalent speed. In short, the speed can be controlled by the subdivision. Furthermore, the subdivision value is inversely proportional to the speed. For example, if the *PWM-Effective-Width* is from 500μs to 2500μs, the subdivision value is 240, the start equivalent position value is 30 and the goal equivalent position value is 60. The variation of the pulse width is equal to "(2500-500)*(60-30)/250=240μs", which would be completed within 240 PWM signal cycles.

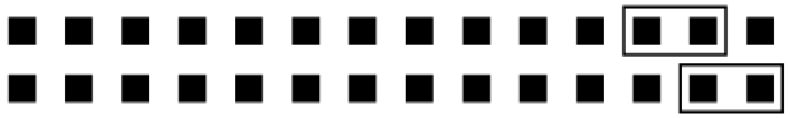
- **What is the difference between the Bulk-Transmission and the Control-Transmission?**

It is simply mentioned in the RS485 communication protocol(**ManShow-RC1** does not integrate RS485, but its UART protocol is the same as the RS485 of **Gicren**'s standard product line). In the servo applications, sometimes we focus on the precision of the pulse width, while we focus on the stability of the communication(any of a valid instruction packet could not be discarded) at other times. Although they are obviously contradictory, the contradiction can become less obvious by improving the processing speed of the MCU and optimizing the code. Taking into account the compatibility of the signal level and the simplicity of the circuit board, we prefer the MCU with operating voltage of 5V and make the above-mentioned contradiction become less obvious by optimizing the code. Of course, all the incoming data can be received by DMA, and the above-mentioned contradiction can be resolved completely, however, most of the MCUs that integrate DMA function work at 3.3V or below. Because the clock of I2C can be stretched, the *Bulk-Transmission* and *Control-Transmission* are only for UART(Note: in the *Bulk-Transmission* mode, the baud rate must be higher than 10000bps). In the circuit board of **ManShow-RC1**, I2C is designed to be the default and fixed communication interface between Arduino UNO R3 and **SoftServo**, and UART is mainly designed to directly combine USB and **SoftServo**(USB<->UART<->SoftServo) to use the **Gicren Tool** for getting the servo's action code(please ensure there is no communication between Arduino UNO R3 and **SoftServo** via I2C or Short circuit between UNO-RST and GND in Expansion-Port) or to be another interface between Arduino UNO R3 and **SoftServo**. Perhaps you still wonder that how to ensure an instruction packet could be correctly received in the *Bulk-Transmission* mode. It is an effective solution that an instruction packet would be sent several times (3 is recommended).

Examples

- **(Single-Servo-Position-Speed-Control) speed controlled by potentiometer, position controlled by two buttons**

  Step1: Ensuring the connection: USB<->UART<->UNO. The corresponding connection of the Expansion-Port is shown below:



(USB-UART-TXD <-> UNO-RXD / USB-UART-RXD <-> UNO-TXD)

  Step2: Servo connected to Channel-1, Potentiometer connected to A0, Button-A connected to D2, Button-B connected to D3.

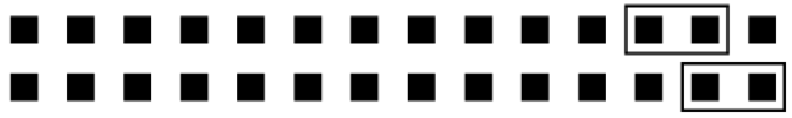
Arduino code(**SingleServoControl**):

```
#include "GI2C_V11.h"
#include <Wire.h>

#define ButtonA            2
#define ButtonB            3
#define PositionA          50
#define PositionB          200
#define Potentiometer    A0

unsigned char Buf[61+1];
GI2CV11 ManShow_RC1(Buf,sizeof(Buf));

void setup()
{
   pinMode(ButtonA,INPUT_PULLUP);
   pinMode(ButtonB,INPUT_PULLUP);
}

void loop()
{
   ManShow_RC1.Read(1,13,48);
   while(1)
   {
      Buf[37]=(unsigned long int)250*analogRead(Potentiometer)/1024;
      if(Buf[37]==0)
```

```
        {
            Buf[37]=1;
        }
        if((digitalRead(ButtonA)==LOW) && (digitalRead(ButtonB)==HIGH))
        {
            Buf[13]=PositionA;
            ManShow_RC1.Write(1,13,48);
        }
        else if((digitalRead(ButtonA)==HIGH) && (digitalRead(ButtonB)==LOW))
        {
            Buf[13]=PositionB;
            ManShow_RC1.Write(1,13,48);
        }
        delay(20);
    }
}
```

- **(Multi-Servo-Position-Speed-Control) 24 servos swing back and forth with different speed**

  Step1: Ensuring the connection: USB<->UART<->UNO. The corresponding connection of the Expansion-Port is shown below:



(USB-UART-TXD <-> UNO-RXD / USB-UART-RXD <-> UNO-TXD)

  Step2: 24 servos connected to each channel(you can also connect one or two servos for test).

  Arduino code(**MultiServoControl**):

```
#include "GI2C_V11.h"
#include <Wire.h>

unsigned char Buf[61+1];
GI2CV11 ManShow_RC1(Buf,sizeof(Buf));

void setup()
{

}

void loop()
{
    unsigned char i;
    unsigned long int j;
    ManShow_RC1.Read(1,13,48);
    for(i=13;i<37;i++)
        Buf[i]=50;
    for(i=37;i<61;i++)
        Buf[i]=(i-36)*10;
    while(1)
    {
        for(j=0;j<2677114440;j++)
        {
            for(i=13;i<37;i++)
            {
                if((j+1)%(i-12)==0)
                {
                    if(Buf[i]==50)
                        Buf[i]=200;
                    else if(Buf[i]==200)
                        Buf[i]=50;
                }
            }
            delay(200);
            ManShow_RC1.Write(1,13,48);
        }
    }
}
```

These two examples are intended to make a simple introduction on how to control the position and speed of servo with the Arduino UNO R3, which also demonstrated that all the servos can be controlled separately. It is very easy to control servos by setting the corresponding registers of each channel at any time. Now, please use the robotic controller(**ManShow-RC1**) and rich Arduino resources to creat your robots!