CAP126

CAPTEUR DE DISTANCE LASER SORTIE RS485

(4-400CM) SEN0492

Introduction

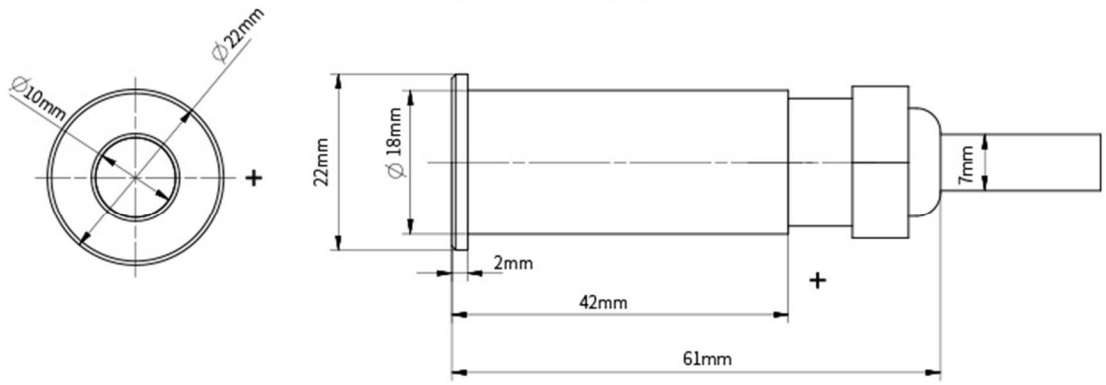
This laser range sensor can be used to detect objects in 4~400cm with ±2cm accuracy. With 3 measurement modes supported, it is well applicable to various detection scenarios like security gate detection, access control systems, security alarm devices, smart trash cans, smart cars or robots for obstacle avoidance.
The sensor metal black shell adopts a threaded barrel design and its probe comes with an optical cover. The integrated design makes the product able to effectively filter optical interference, waterproof and shockproof. The sensor uses invisible laser and there is a voltage regulator circuit inside the module. It operates at 5V-36V and employs industrial 485 chip, which supports Modbus-RTU protocol and works well with industrial 485 devices. The sensor has an alarm output line that will be triggered to output Low steadily when the measurement distance is smaller than the user-set threshold.

`Note: The data will be unstable when measuring black objects.`

Specification

- Power Supply Voltage: 5-36 V
- Measuring Distance: 4-400 cm
- Measurement Accuracy: ±2 cm
- Launch Angle: 39.6°
- Receiving Angle: 36.5°
- Working Current: <38mA
- Communication Interface: RS485
- Communication Protocol: Modbus-RTU
- Waterproof Rating: IP67
- Baud Rate: 2400-921600 optional 115200 (default)
- Frequency: 20Hz (default)
- Working Temperature: -20~70°C
- Size: 21.5x21x8 mm / 0.85×0.83×0.31 inch

Dimension Figure



Board Overview

Label	Name	Description
Red	VCC	5-36V power supply
Green	B	485 line B
Yellow	A	485 line A
Black	GND	Power ground wire
White	ALARM	Police connection line

Communication protocol

The sensor adopts the industry standard Modbus protocol, the specific read and write format is as follows: Modbus communication, the command number is divided into two types, read command and write command, 0x03 (read command) reads the corresponding register data, 0x06 (write command) write data to the corresponding register.

Host sending frame (HEX)

Slave address	Function code	Register address high bit	Register address low bit	Read
0x50	0x03	RegH	RegL	LenH

The module address is 0x50 (default), the read command is 0x03, the register 0x34 (measurement distance), and the length is one bit. Command: 50 03 00 34 00 01 C8 45 Slave response frame (HEX)

Slave address	Function code	Data length	Data bit 1	D
0x50	0x03	LenH	DataH	D

The module address is 0x50, the read command is 0x03, and the length is 2 bits. Example: Read the measured distance Send command: 50 03 00 34 00 01 c8 45 Receive data: 50 03 02 07 0B 06 7F

Data analysis: 0x50 is Modbus address, 0x03 read command, 0x02 data length, 0x07 0x0B measurement data corresponding to 0x070B is decimal 1803, measurement distance is 1803mm, 0x06 0x7F is CRC check bit.

Register table

Register name	**Register address**	**Command**	**Description**
System recovery	0x00	MODADDR 06 00 00 00 01 CRCH CRCL	Write 0x01, the sensor restores the default setting
Alarm threshold	0x02	MODADDR 06 00 03MH ML CRCH CRCL	MH alarm threshold high bit, ML alarm threshold low bit, threshold setting range 40~4000mm
Baud rate setting	0x04	MODADDR 06 00 04 00 00 CRCH CRCL	Write 0x00, baud rate 2400
		MODADDR 06 00 04 00 01 CRCH CRCL	Write 0x01, baud rate 4800
		MODADDR 06 00 04 00 02 CRCH CRCL	Write 0x02, the baud rate is 9600
		MODADDR 06 00 04 00 03 CRCH CRCL	Write 0x03, the baud rate is 19200
		MODADDR 06 00 04 00 04 CRCH CRCL	Write 0x04, the baud rate is 38400
		MODADDR 06 00 04 00 05 CRCH CRCL	Write 0x05, baud rate 57600
		MODADDR 06 00 04 00 06 CRCH CRCL	Write 0x06, baud rate 115200
		MODADDR 06 00 04 00 07 CRCH CRCL	Write 0x07, baud rate 230400

Register name	Register address	Command	Description
		MODADDR 06 00 04 00 08 CRCH CRCL	Write 0x08, baud rate 460800
		MODADDR 06 00 04 00 09 CRCH CRCL	Write 0x09, the baud rate is 921600
Timing preset time (not recommended to modify, default 200MS)	0x07	MODADDR 06 00 07 TIMEBUDGETH	TIMEBUDGET: 20-1000 milliseconds, can be changed to 0x0014-0x03e8
Measurement interval (not recommended to modify, default 50MS)	0x08	MODADDR 06 00 08 PERIODH PERIODL CRCH CRCL	PERIOD: 1-1000 milliseconds, can be changed to 0x0001-0x03e8
ID setting	0x1A	MODADDR 06 00 1a 00 MODADDRL CRCH CRCL	Can write 0x00~0xFE
Measurement data	0x34	MODADDR 03 00 34 00 01 CRCH CRCL	Read, the upper 8 bits of the distance to the lower 8 bits of the distance
Output state	0x35	MODADDR 03 00 35 00 01 CRCH CRCL	Read: 0x07, sensor No Update
			Read: 0x00, sensor Range Valid
			Read: 0x01, sensor Sigma Fail
			Read: 0x02, sensor Signal Fail
			Read: 0x03, sensor Min Range Fail
			Read: 0x04, sensor Phase Fail

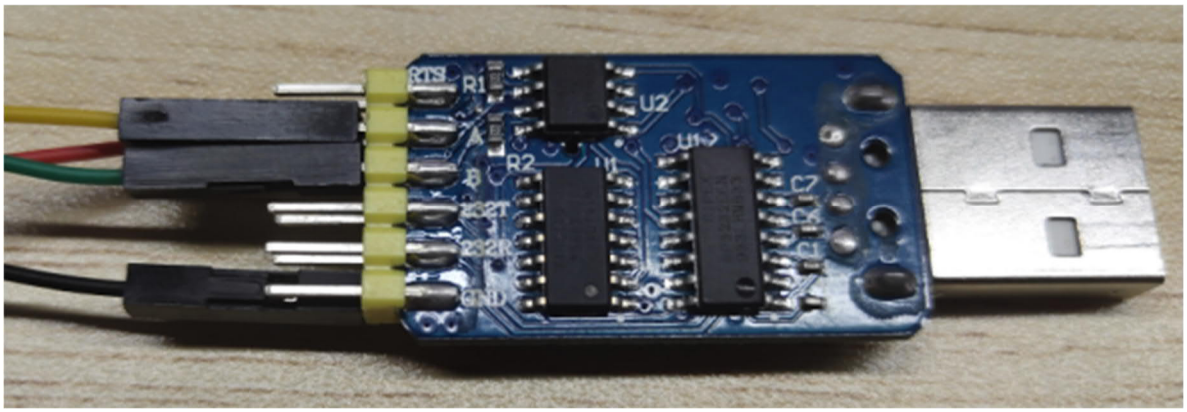
Register name	Register address	Command	Description
			Read: 0x05, Sensor Hardware Fail
Measurement mode	0x36	MODADDR 06 00 36 00 01 CRCH CRCL	Write 0x01, short distance (up to 1.3m, better environmental immunity)
		MODADDR 06 00 36 00 02 CRCH CRCL	Write 0x02, middle distance (up to 3 meters)
		MODADDR 06 00 36 00 03 CRCH CRCL	Write 0x03, long distance mode (up to 4 meters)
Calibration mode	0x37	MODADDR 06 00 37 00 04 CRCH CRCL	Write 0x04 to enter the calibration state
		MODADDR 03 00 37 00 01 CRCH CRCL	Read: 0x01, start calibration
		MODADDR 03 00 37 00 02 CRCH CRCL	Read: 0x02, calibration failed
		MODADDR 03 00 37 00 03 CRCH CRCL	Read: 0x03, calibration is complete

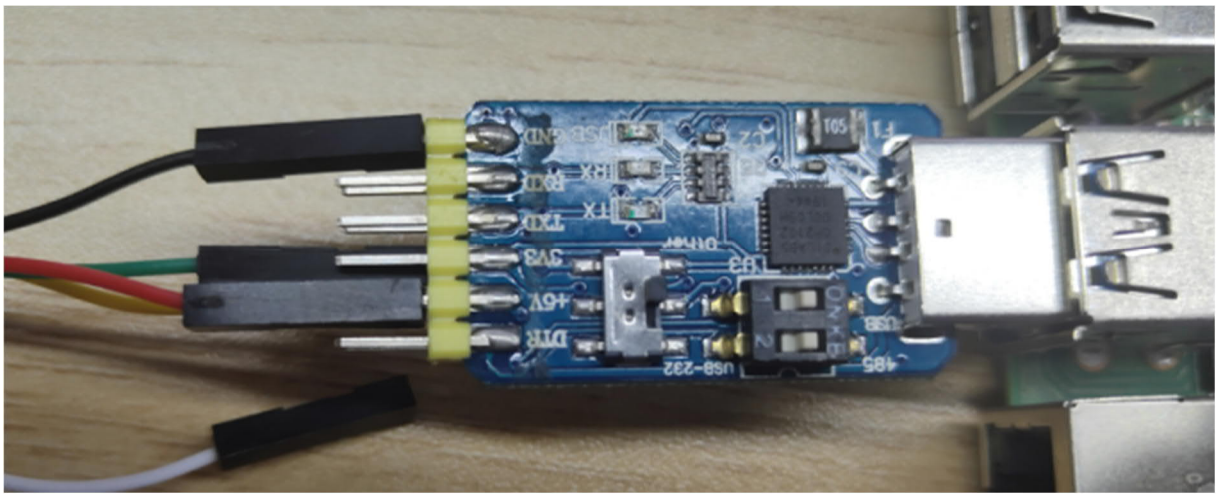
Tutorial for Raspberry Pi

Requirements

- **Hardware**
  - Raspberry Pi x 1
  - 6-in-1 multi-function to serial port module or USB to RS485 module x1
  - Laser ranging sensor RS485 (4m) x1

1. Connection Diagram



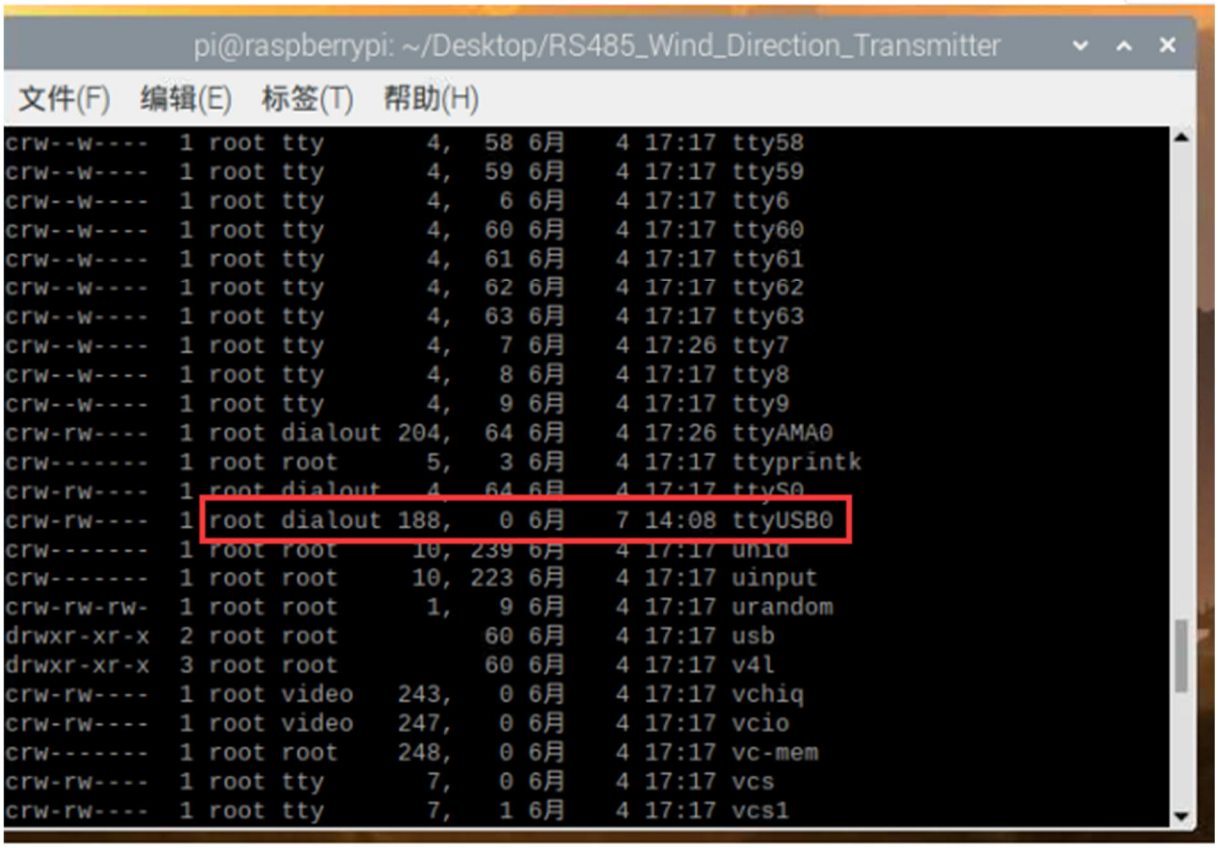


2. Check usb device

Type in the terminal

```
sudo ls -l /dev
```

Find the USB device that has just been connected to the Raspberry Pi (every time the USB device is connected to the Raspberry Pi, the device port will change, so you need to check the actual port each time when you connect to the Raspberry Pi)



3. Install the wiringpi library

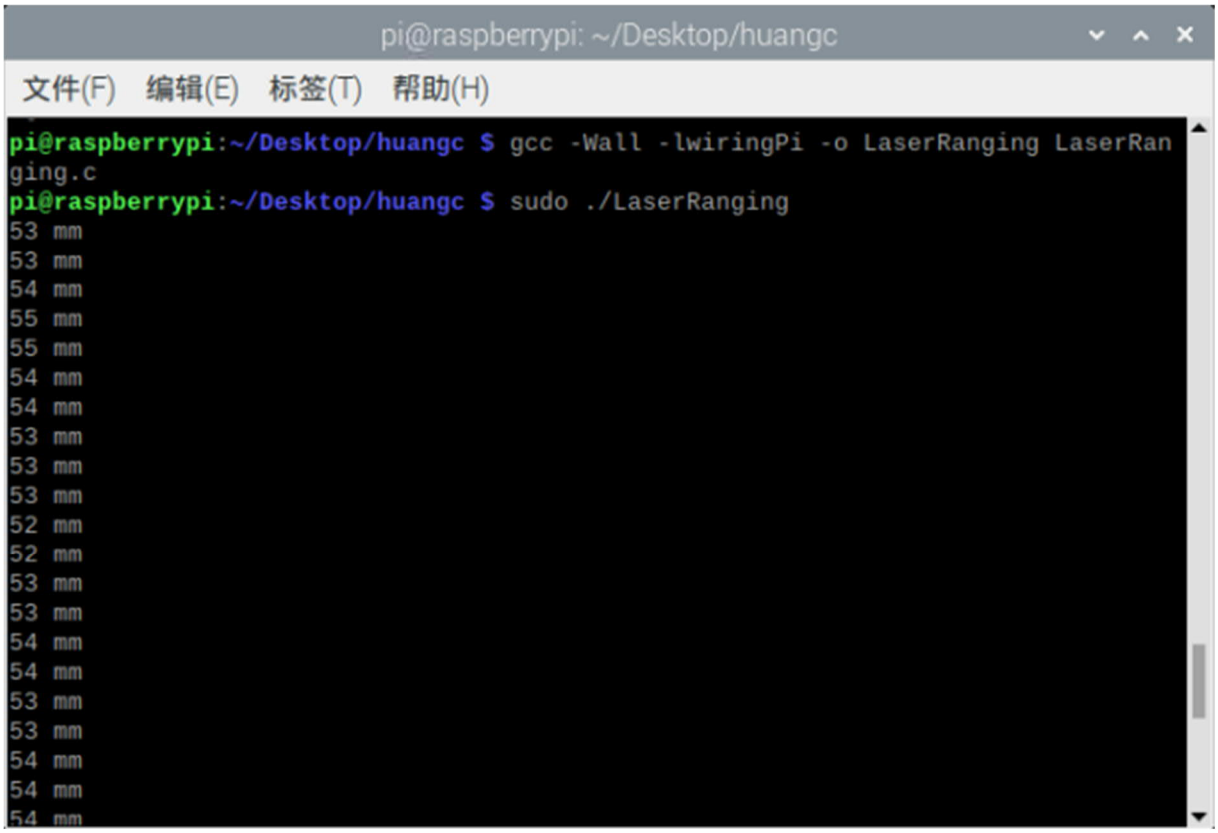
```
cd /tmp
wget https://project-downloads.drogon.net/wiringpi-latest.deb //Download wiringpi library
sudo dpkg -i wiringpi-latest.deb //Install wiringpi library
```

4. Compile and run the sample code

Create a new folder on the desktop, create a new LaserRanging.c file in the folder, copy the code in and save it, then use the terminal to open the folder where the program is located, compile and run

```
gcc -Wall -lwiringPi -o LaserRanging LaserRanging.c
```

Then you can see the accurate measured distance value



Sample Code

```
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>

#include <wiringPi.h>

#include <wiringSerial.h>

int recData(unsigned char* buf);
unsigned int CRC16_2(unsigned char *buf, int len);

int fd;

unsigned char Data[8] = {0};

int main()
{
    if ((fd = serialOpen("/dev/ttyUSB0", 115200)) < 0) {     //You can change usb
device port in line with the actual conditions here.
        fprintf(stderr, "Unable to open serial device: %s\n", strerror(errno));
        return 0;
    }
    while(1){
        delay(100);
        printf("%d mm\n",recData(Data));
    }
    return 1;
```

```
}

int recData(unsigned char* buf)
{
    char ret = 0;
    int jl=0;
    long curr = millis();
    char ch = 0;
    unsigned char COM[8]={0x50, 0x03, 0x00, 0x34, 0x00, 0x01, 0xC8, 0x45};
    write(fd, COM, 8);
    while(!ret){
        if (millis() - curr > 1000){
            //write(fd, COM, 8);
            curr = millis();
            printf("OK\n");
        }

        if(serialDataAvail(fd) > 0){
            delay(10);
            if (read(fd, &ch, 1) == 1){
                if(ch == 0x50){
                    buf[0] = ch;
                    if (read(fd, &ch, 1) == 1){
                        if(ch == 0x03){
                            buf[1] = ch;
                            if (read(fd, &ch, 1) == 1){
                                if(ch == 0x02){
                                    buf[2] = ch;
                                    if (read(fd, &buf[3], 4) == 4){
                                        // for(int i=0; i<7; i++){
                                        //     if(buf[i] < 0x10){
                                        //         printf("0");
                                        //     }
                                        //     printf("%x ", buf[i]);
                                        // }
                                        // printf("\n");
                                        // printf("%x\n", CRC16_2(buf, 5));
                                        // printf("%x\n", buf[5]*256+buf[6]);
//Used to view the raw data of the distance measured by the sensor.
                                        if(CRC16_2(buf, 5) == (buf[5] * 256 +
buf[6])){
                                            jl = buf[3]*256+buf[4];
                                            ret = 1;
                                        }

                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    return jl;
}

unsigned int CRC16_2(unsigned char *buf, int len)
```

```
{
  unsigned int crc = 0xFFFF;
  for (int pos = 0; pos < len; pos++)
  {
    crc ^= (unsigned int)buf[pos];
    for (int i = 8; i != 0; i--)
    {
      if ((crc & 0x0001) != 0)
      {
        crc >>= 1;
        crc ^= 0xA001;
      }
      else
      {
        crc >>= 1;
      }
    }
  }

  crc = ((crc & 0x00ff) << 8) | ((crc & 0xff00) >> 8);
  return crc;
}
```