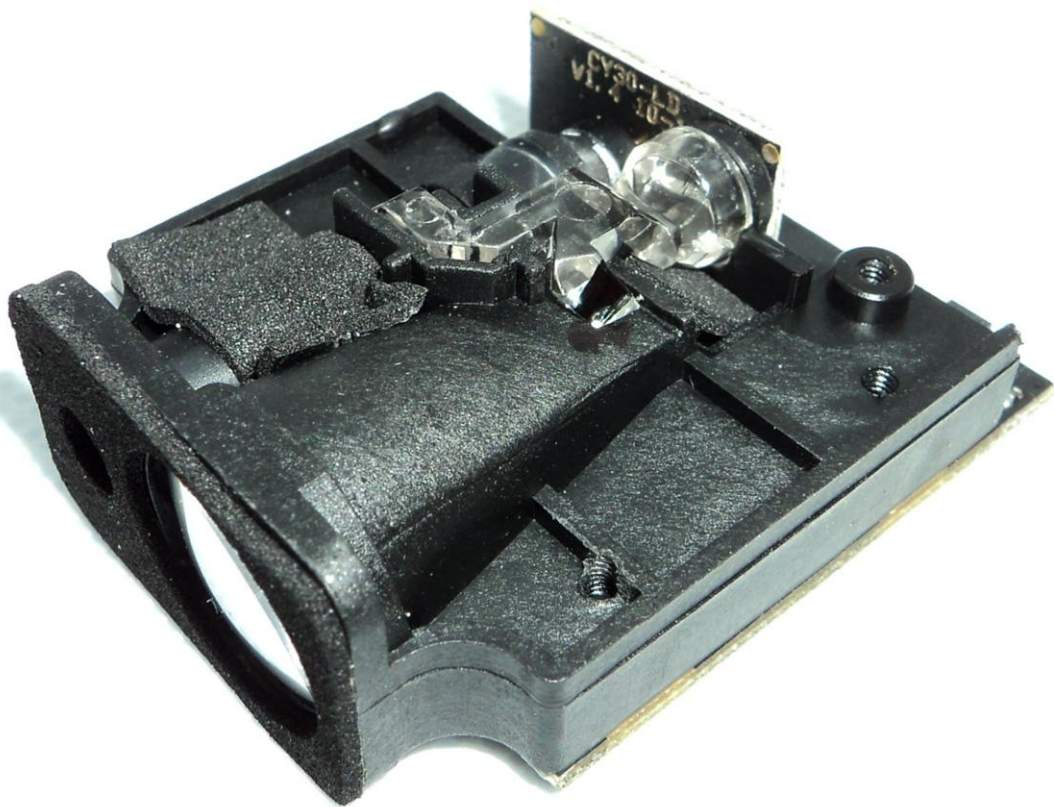
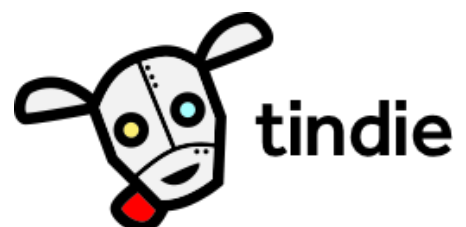


Laser Rangefinder Module



Available on [Tindie.com](https://www.tindie.com)



Technical Parameters:

Measuring range	40 m
Resolution	0.01 mm (0.00001 m)
Measurement accuracy (standard deviation)	± 2 .mm (within 10 m).More than 10m the formula is $\pm 2 + 0.05 * (D-10)$, D is the distance
Distance Unit	meter
Laser Type	620-690 nm
Laser Class	2nd grade, <1mW (secondary safety)
Single measurement time	0.25s
Spot diameter	6 mm @ 10 m
Working temperature	0 ~ + 40 °C
Storage temperature	-20 ~ + 60 °C
Weight	60 g
Dimensions	4.8 cm X 3.7 cm X 1.8 cm
Supply Voltage	DC 3v to 3.3v max
I/O voltage	TTL level compatible 3v-5v voltage

Note:

- In harsh environments such as: the sun is too strong, the ambient temperature fluctuations, the reflecting surface is relatively weak effect, system voltage is too high or too low, if the measure distance is close, the result may be not be correct , we recommend to use a target plate in this situation.
- It's very important to power up the module with a stable power supply system & any supply voltage more than 3.3v will damage the module
- *The Rx & Tx pins of the module should be connected through a serial current-limiting resistance (1k at least) to work correctly.*

Communication Interface:

This module uses the TTL Serial protocol UART (Rx/Tx) for data exchange, & its communication rates can up to 115K (baud rate 115200)

*Power up on the module, but don't measure & you will see that the laser turn off automatically after the timeout
You can use the appropriate instructions to set the measurement mode*

Instructions

Instruction	Meaning	Module Answer	Notes
\$00022123&	Start a single measurement	ok(Confirm)	Measurement data after confirmation
\$0003260130&	Turn on the laser light	Confirm + repeat command	
\$00022426&	Open the continuous measurement	ok(Confirm)	Measurement data after confirmation
\$0003260029&	Stop continuous measurement	Confirm + repeat command	
\$00022123&	Turn off the module	ok(Confirm)	The module don't have a sleep mode, if you need to close the laser, use this instruction

Module Answer	Meaning
\$00023335&	Message confirmation (OK)
xxx0001643&	Last 7 bits (No signal error)
xxx0001542&	Last 7 bits (The measured distance is too close/ Error)

How to use this module for the first time

- 1- Power up the module by connecting it to your computer or laptop via any 3.3v USB/TTL converter
- 2- Open your Serial terminal & select your device COM & Baud-Rate (115200)
- 3- Turn on the laser light by this instruction **\$0003260130&** to target easily your object or any surface that you like to measure his distance ... or just skip it if you like
- 4- Send the instruction **\$00022123&** to take a single measurement of the targeted object or surface.

You will receive automatically a reply from the module including the message confirmation + the measurement information. **See the Single Measurement Mode & Data Analysis** for more information.
Note that the module will stop automatically.

- 5- To use the continuous measurement mode, just send this instruction **\$00022426&** to the module & you will receive a multiple & fast measurement information of your targets. **See the Continuous Measurement Mode & Data Analysis** for more information.

You can stop measurement by sending this instruction **\$0003260029&**

- 6- If there are no need to take any measurement, you can close & shutdown the module with this instruction **\$00022123&**

Single Measurement Mode & Data Analysis

A single measurement mode is required to ensure that the laser beam is on the target surface, and if not sure the laser is pointed to the target surface before measurement, please execute the instruction of “opening laser”.

Host send => \$00022123&

Module reply => \$00023335& + Measurement Data

The Data Analysis in Single Measurement Mode:

The result of any correct single measurement data will looks like this example

\$00023335&\$0006210000008916&

The measurement is 0.08916 Metter

- **Blue total 8 bits:** response confirmation of the module in a single measurement mode.
- **Green & Red total 7 bits:** indicate the distance, the first 2 bits indicate the integer part & the last 5 bits indicate decimal part
- The other black bits are unused/reserved

Error Codes in Single Measurement Mode:

Error Code	Reason	Error Response Time
\$00023335&\$0006210000001542&	Distance too short	About 5 seconds from sending instructions to returning an error code
\$00023335&\$0006210000001643&	No echo	About 5 seconds from sending instructions to returning an error code
\$00023335&\$0006210000001744&	reflection is too strong	About 5 seconds from sending instructions to returning an error code
\$00023335&\$0006210000001845&	ambient light is too strong	About 5 seconds from sending instructions to returning an error code

Continuous Measurement Mode & Data Analysis

In continuous measurement mode, the host machine sends a continuous measurement instruction the module & then takes a successive measurement.

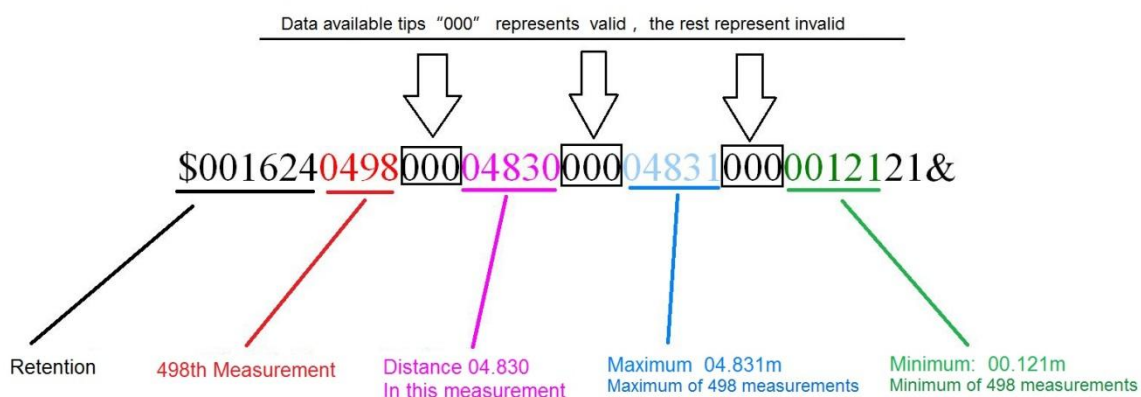
The data returned by the module contains the measured distance, maximum, minimum, etc., and in front of each value there is 3 characters "000" used to mark if the data is valid. If there is an error (blocking the laser emission port, the lens, or too close...) ,it output an error code & set the measurement count to 9999, and terminate the measurement.

In continuous measurement mode, the laser will be closed and re-opened every five measurements to reduce the CPU load and increase measurement speed.

This mode may be used to sweep a plane or three-dimensional surface slowly, such as a wall, thereby obtaining the maximum and minimum value, can also be used for real-time distance measurement.

The Data Analysis in Continuous Measurement Mode:

The result of any correct measurement data will looks like this examples



- **Black bits: Reserved for the module**
- **Red 4 bits: Counting the number of measurements. Last measurement error or termination measurement, the measurement number counter is equal to 9999. This section data can be ignored & it's used for continuous uninterrupted Ranging.**
- **Pink 5 bits: indicate the distance, the first 2 bits indicate the integer part, the last 3 bits indicate decimal part.**

When the Host machine send \$00022426&, the module reply \$00023335& + returns ranging data.

Note that the first measurement data return will bring a response at the front that can be directly discarded.

Data analysis Example:

\$001624000100004855000048550000485550&

1st time/ Measurement=04.855 m/Maximum=04.855m/Minimum=04.855m
(Three values of the first measurement are the same)

\$001624001000000136000048550000413627&

10th time/Measurement=00.136m/Maximum=04.855m/Minimum=04.136m

\$001624049800004830000048310000012121&

498th time/Measurement=04.830m/Maximum=04.831m/Minimum=00.121m

\$001624999900004830000048310000012117&

The last time, the date can be discarded.

Error Codes in the Continuous Measurement Mode:

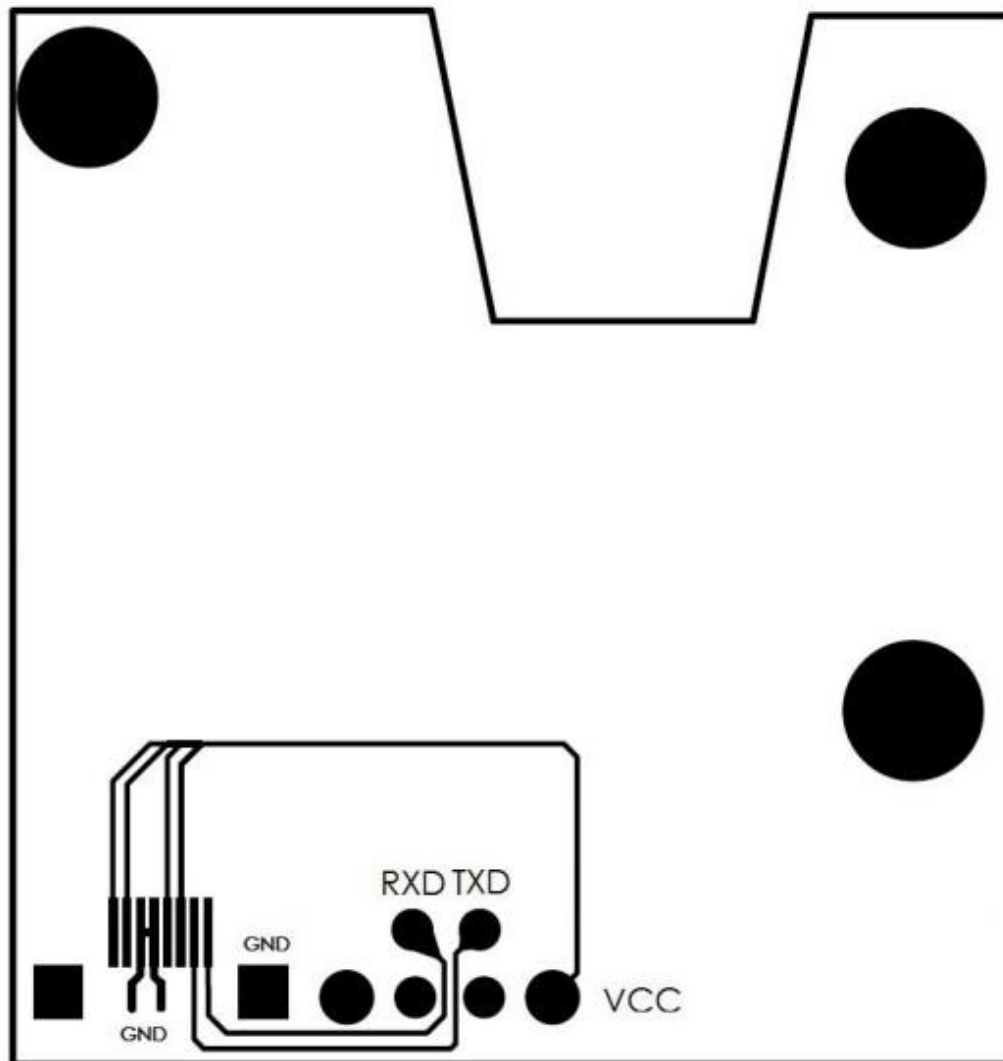
\$001624000100118297001182970011829711&

The first measurement data is invalid, the error code 18 value

Error code	Reason	Error Response Time
\$001624999900000015000 000000000000053&	Distance too short	About 5 seconds from sending instructions to returning an error code
\$001624999900000016000 000000000000054&	No echo	About 5 seconds from sending instructions to returning an error code
\$001624000100118297001 182970011829711&	Ranging error	About 5 seconds from sending instructions to returning an error code

Appendix

Dimensions and interfaces:



Port	Voltage	Function	Note
VCC	3-3.3V	Module power supply	More than 3.3V, the module will burn!!!!
GND		Modules Power ground	
TXD	3-5V	Module sends data pin	It can be connected to a 5v microcontroller I/O trough a serial resistance of 1k (current limiting resistor) You can use it directly with a 3.3v microcontroller I/O
RXD	3-5V	Module receives data pin	It can be connected to a 5v microcontroller I/O trough a serial resistance of 1k (current limiting resistor) You can use it directly with a 3.3v microcontroller I/O
FPC			0.5mm 8p clamshell

Arduino Program

```
#include <Wire.h>
#include <Arduino.h>
#include <SoftwareSerial.h>
//ARDUNIO Serial control command

#define LASER_OPEN "O"//Open the module
#define MEASURE_ONE "D"//Single measurement
#define MEASURE_CONTINUE "C"//Continuous measurement
#define STOP_MEASURE "S"//Stop measuring
// software serial : TX = digital pin 7, RX = digital pin 6
SoftwareSerial portOne(7, 6);
//Measuring state machine status
#define STEP_LASER_OPEN 0//Open the module
#define STEP_MEASURE_ONE 1//Single measurement
#define STEP_MEASURE_CONTINUE 2//Continuous measurement
#define STEP_STOP_MEASURE 3//Stop measuring

#define MEASURE_CONTINUE_N 200//Continuous measurement

#define SUCCESS 0//measurement
#define FAIL -1//measurement

unsigned char measure_continu_cnt=0;
unsigned char measure_continue_flag=0;
//Function declaration
String ReadData(char Length);
String Serial_One_ReadData(char Length);
void Measure(char MEASURE_COMMEND);//distance measurement
String LaserOpen();
String MeasureOne();
String MeasureContinue();
void measure_continue_data();
String StopMeasure();

void setup() {
    // put your setup code here, to run once:
    Serial.begin(9600);//Serial debug baud rate
    while (!Serial) {
        ; // wait for serial port to connect. Needed for native USB port only
    }
    // Start both software serial ports
    portOne.begin(115200); //Laser ranging module baud rate
}
```



```

void loop() {
    String temp="";
    temp= ReadData();
    if(temp == LASER_OPEN)
        Measure(STEP_LASER_OPEN);
    else if(temp == MEASURE_ONE)
        Measure(STEP_MEASURE_ONE);
    else if(temp == MEASURE_CONTINUE)
        Measure(STEP_MEASURE_CONTINUE);
    else if(temp == STOP_MEASURE)
        Measure(STEP_STOP_MEASURE);
    else if(temp != "")
        Serial.print("UNDEFINED COMMEND\r\n");
    if(measure_continue_flag==1)
    {
        measure_continue_data();
    }
    ;
}

```

```

void Measure(char STEP_MEASURE_COMMEND)//distance measurement
{
    String temp="";
    switch(STEP_MEASURE_COMMEND)
    {
        case STEP_LASER_OPEN:
            temp = LaserOpen();
            if(temp!= "-1")
            {
                Serial.print("LASER OPEN SUCCEED\r\n");
            }
            else
            {
                Serial.print("LASER OPEN FAILD\r\n");
            }
            break;
        case STEP_MEASURE_ONE:
            temp = MeasureOne();
            if(temp!= "-1")
            {
                if(temp.indexOf("&")==9 | temp.lastIndexOf("$")==10)
                {
                    temp=temp.substring(20,22)+"."+temp.substring(22,27);
                }
            }
            else
            {

```

```

        temp=temp.substring(10,12)+ "." +temp.substring(12,17);
    }
    temp="distanc="+temp+"m"+"r\n";
    Serial.print(temp);
}
else
{
    Serial.print("MEASURE_ONE FAILD\r\n");
}
break;
case STEP_MEASURE_CONTINUE:
    Serial.print("MEASURE_CONTINUE START\r\n");
    temp=MeasureContinue();
    measure_continu_cnt=0;
    measure_continue_flag=1;
    if(temp!= "-1")
    {
        if(temp.indexOf("&")==9 | temp.lastIndexOf("$")==10)
        {
            temp=temp.substring(24,26)+ "." +temp.substring(26,29);
        }
        else
        {
            temp=temp.substring(14,16)+ "." +temp.substring(16,19);
        }
        temp="distanc="+temp+"m"+"r\n";
        Serial.print(temp);
    }
    else
    {
        Serial.print("MEASURE_ONE FAILD\r\n");
    }
    break;
case STEP_STOP_MEASURE:
    temp = StopMeasure();
    measure_continu_cnt=0;
    measure_continue_flag=0;
    Serial.print("MEASURE STOP\r\n");
    break;
default:
    break;
}
}

//Turn on the laser pointer
String LaserOpen()
{

```

```

String return_data="";
unsigned char temp=0;
portOne.listen();

if (portOne.isListening()) {
    portOne.flush();
    portOne.print("$0003260130&");
    return_data = Serial_One_ReadData(22);
    return return_data;//Turn on the laser module
}
}

//Single range measurement
String MeasureOne()
{
    String return_data="";
    unsigned char temp=0;
    portOne.listen();
    if (portOne.isListening()) {
        portOne.flush();
        portOne.print("$00022123&");
        return_data = Serial_One_ReadData(28);
        return return_data;
    }
}

//Continuous ranging
String MeasureContinue()
{
    String return_data="";
    unsigned char temp=0;
    portOne.listen();

    if (portOne.isListening()) {
        portOne.flush();
        portOne.print("$00022426&");
        return_data = Serial_One_ReadData(48);
        return return_data;
    }
}

void measure_continue_data()
{
    String temp="";
    temp=Serial_One_ReadData(38);
    measure_continu_cnt++;
    if(temp.substring(17,19)=="15" | temp.substring(17,19)=="16" | temp.substring(17,19)=="")

```

```

        {
            Serial.print( measure_continu_cnt);
            Serial.print("ERROR\r\n");
            temp=MeasureContinue();
            temp=temp.substring(24,26)+ "." +temp.substring(26,29);;
            Serial.print( measure_continu_cnt);
            temp="  distanc="+temp+"m"+"r\n";
        }
        else
        {
            temp=temp.substring(14,16)+ "." +temp.substring(16,19);;
            Serial.print( measure_continu_cnt);
            temp="  distanc="+temp+"m"+"r\n";
        }
        Serial.print(temp);
    }
    //Stop measurement
    String StopMeasure()
    {
        String return_data="";
        unsigned char temp=0;

        portOne.listen();

        if (portOne.isListening()) {
            portOne.flush();
            portOne.print("$0003260029&");
            return_data = Serial_One_ReadData(22);
            return  return_data;
        }
    }

    //Read serial port to return data
    String ReadData()
    {
        int wait_time=0;
        String return_data="";
        do
        {
            delay(1);
            wait_time++;
            while(Serial.available()>0)//serial data read
            {
                char CharRead=Serial.read();
                if(CharRead!=10&&CharRead!=13)
                {
                    return_data +=CharRead;
                }
            }
        }
    }

```

```

    }
    wait_time=0;
}
}
while(wait_time<500); //Wait for 1 second, no data is returned, the read is finished.
return return_data;
}

```

//Read serial port to return data

String Serial_One_ReadData(char Length)

```

{
    int wait_time=0;
    String return_data="";
    do
    {
        delay(1);
        wait_time++;
        while( portOne.available(>0)//serial data read
        {
            char CharRead= portOne.read();
            if(CharRead!=10&&CharRead!=13)
            {
                return_data +=CharRead;
            }
            wait_time=0;
            if(return_data.length()>=Length)
            {
                wait_time=6000;
            }
        }
    }
    while(wait_time<2000); //Wait for 1 second, no data is returned, the read is finished.
    unsigned char last_byte=(unsigned char)return_data.charAt(return_data.length()-1);
    if( wait_time == 6000)
    {
    }
    else
    {
        return_data="-1";
    }
    return return_data;
}

```