WitMotion Standard Protocol-PYTHON-SDK Quick Start

Protocol Induction

WITMOTION standard protocol: referred to as WITMOTION protocol, is the protocol used by WITMOTION smart sensors, usually TTL or 232 level sensors use WITMOTION protocol; the protocol stipulates that the sensor returns data packets starting with 55, and the PC software sends FF AA. Data packets beginning with AA;

Routine introduction

This routine introduces how to use python to develop the pc software to connect the WITMOTION protocol sensor, receive sensor data and communicate with the sensor;

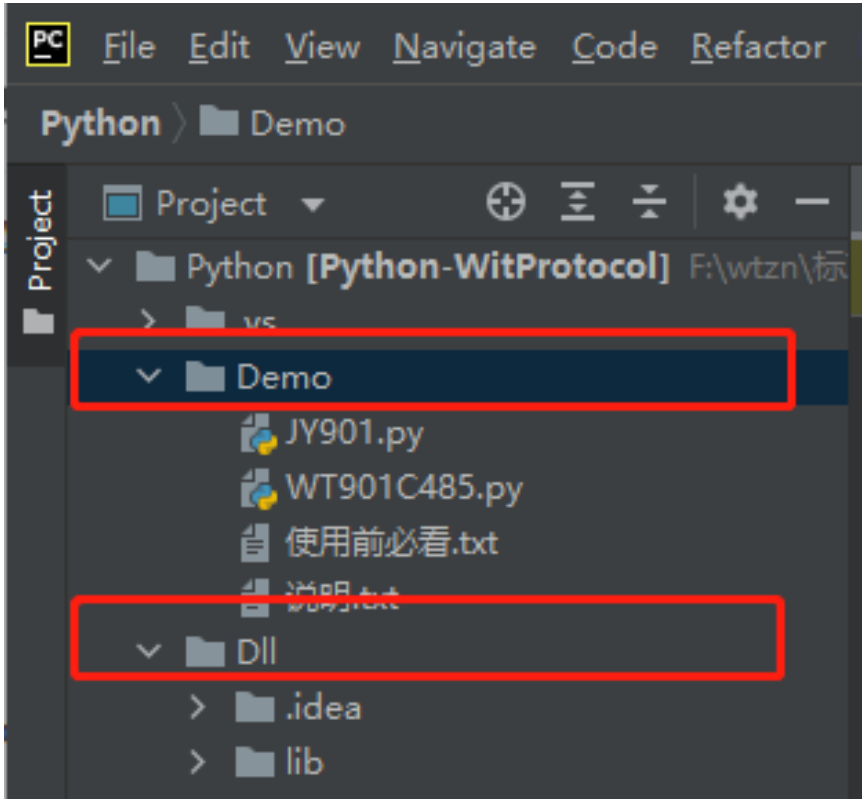
Please read the relevant sensor manual before viewing this routine to understand the protocol used by the sensor and the basic functions of the sensor

Routine directory

The routine project directory is as follows

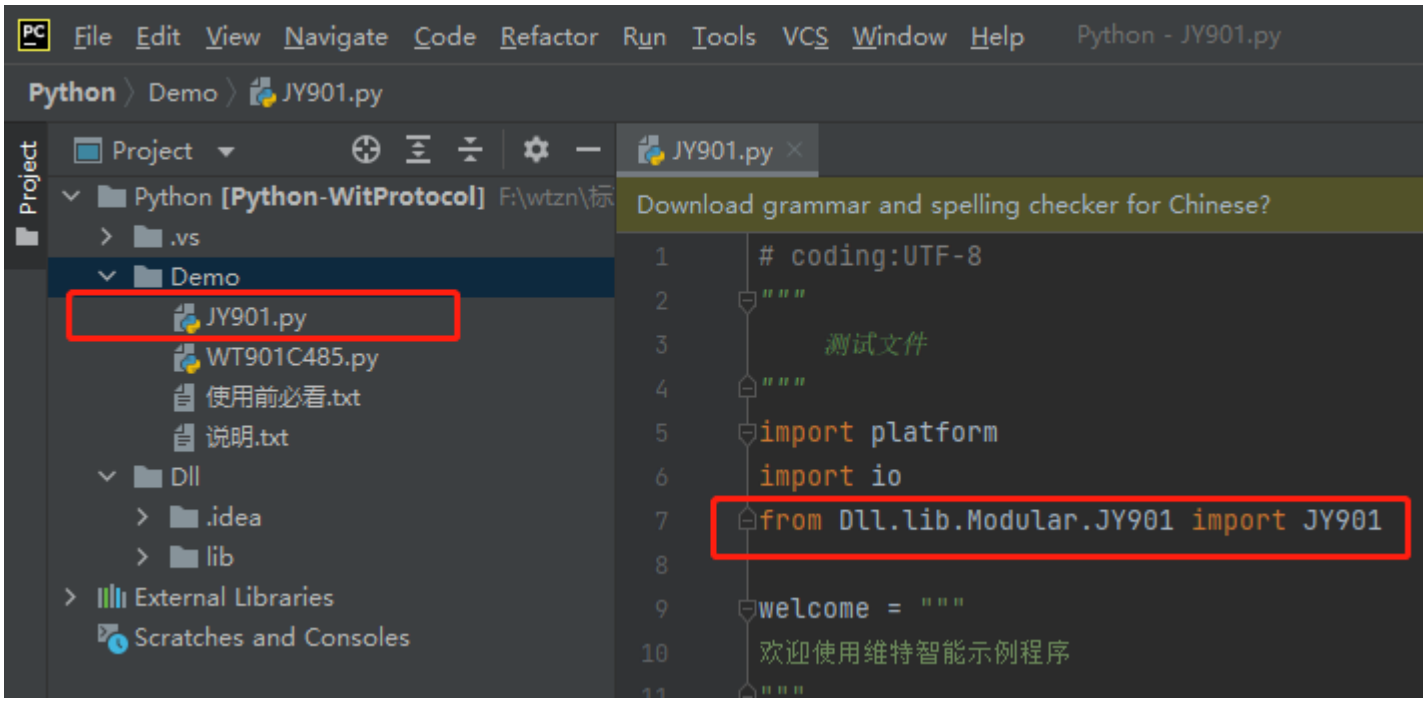
Dll: The dependency file of the project.

Demo: JY901.py in the routine is the protocol used by the WITMOTION sensor


File Edit View Navigate Code Refactor
Python Demo
Project
Project
Python [Python-WitProtocol] F:\wtzn\标
Demo
JY901.py
WT901C485.py
使用前必看.txt
Dll
.idea
lib

Import dependency files

The routine project introduces dependency files as follows


File Edit View Navigate Code Refactor Run Tools VCS Window Help
Python - JY901.py
Python 〉 Demo 〉 JY901.py
Project
Python [Python-WitProtocol] F:\wtzn\标
.vs
Demo
JY901.py
WT901C485.py
使用前必看.txt
说明.txt
Dll
.idea
lib
External Libraries
Scratches and Consoles
JY901.py
Download grammar and spelling checker for Chinese?
coding:UTF-8
"""
测试文件
"""
import platform
import io
from Dll.lib.Modular.JY901 import JY901
welcome = """
欢迎使用维特智能示例程序
"""

Initialize the device

The routine works as follows

```
if __name__ == '__main__':
    print(welcome)
    # initialize the device
    JY901 = JY901()
    if platform.system().lower() == 'linux':
        PortName = "/dev/ttyUSB0"
    else:
        PortName = "COM11"
    Baudrate = 9600
    # set serial port
```

```
JY901.SetPortName(PortName)
# set baud rate
JY901.SetBaudrate(Baudrate)
# open serial port
JY901.Open()
if JY901.IsOpen():
    print("Device opened successfully")

    # Add up calibration
    AppliedCalibration(JY901)

    # Magnetic field calibration
    # StartFieldCalibration(JY901)
```

```
# read 0x03 register

# wait time

waitTime = 200

# Send read command and wait for sensor to return data

IsReadReg(JY901, 0X03, waitTime)

# The following line is equivalent to the above. It is recommended to use the above

# JY901.SendProtocolData([0xff, 0xaa, 0x27, 0x03, 0x00], waitTime)

print("0x03 read result:" + str(JY901.GetDeviceData("0x03")))


# Write the corresponding value of the register: write register 0x03 to 0x06

JY901.SendWriteReg(0x03, 0x06)

# save the value of the register
```

```
JY901.SaveReg()

# The following line is equivalent to the above. It is recommended to use the above

# JY901.SendProtocolData([0xff, 0xaa, 0x03, 0x06, 0x00], waitTime)

# save the value of the register

# JY901.SendProtocolData([0xff, 0xaa, 0x00, 0x00, 0x00], waitTime)


# Bind receiving events to record data events

JY901.AddOnRecord(JY901_OnRecord)


input()


# close the device

JY901.Close()
```

```
        # remove event
        JY901.RemoveOnRecord(JY901_OnRecord)
    else:
        print("Failed to open the device")
```

Turn on the device

The JY901 object represents the JY901 device in the program, and you can communicate with the device through it; when opening the device, you need to specify the serial port number and baud rate of the sensor, and call the JY901.Open() method after specifying it.

Turn off the device

Close the device and call the JY901.Close() method.

Receive sensor data

Get data

The JY901 object will automatically solve the sensor data and save it on itself. The sensor data can be obtained through the JY901.GetDeviceData() method. JY901.GetDeviceData() needs to pass in a key to get sensor data.

```
def JY901_OnRecord(deviceModel):
    """
    This is called when sensor data is refreshed and you can log data here
    :param deviceModel:
    :return:
    """
    builder = io.StringIO()
    # device name
    builder.write(deviceModel.deviceName + "\n")
    # chip time
    builder.write("Chiptime:" + str(deviceModel.GetDeviceData("Chiptime"))+"\t")
    # temperature
    builder.write("Temperature:" + str(deviceModel.GetDeviceData("Temperature"))+"\n")
```

```
# acceleration
builder.write("AccX:" + str(deviceModel.GetDeviceData("AccX"))+"g \t")
builder.write("AccY:" + str(deviceModel.GetDeviceData("AccY"))+"g \t")
builder.write("AccZ:" + str(deviceModel.GetDeviceData("AccZ"))+"g \n")
# angular velocity
builder.write("GyroX:" + str(deviceModel.GetDeviceData("GyroX"))+"°/s \t")
builder.write("GyroY:" + str(deviceModel.GetDeviceData("GyroY"))+"°/s \t")
builder.write("GyroZ:" + str(deviceModel.GetDeviceData("GyroZ"))+"°/s \n")
# angle
builder.write("AngleX:" + str(deviceModel.GetDeviceData("AngleX"))+"° \t")
builder.write("AngleY:" + str(deviceModel.GetDeviceData("AngleY"))+"° \t")
builder.write("AngleZ:" + str(deviceModel.GetDeviceData("AngleZ"))+"° \n")
# Magnetic field
```

```
builder.write("MagX:" + str(deviceModel.GetDeviceData("MagX"))+"uT \t")

builder.write("MagY:" + str(deviceModel.GetDeviceData("MagY"))+"uT \t")

builder.write("MagZ:" + str(deviceModel.GetDeviceData("MagZ"))+"uT \n")

# latitude and longitude

builder.write("Lon:" + str(deviceModel.GetDeviceData("Lon"))+"\t")

builder.write("Lat:" + str(deviceModel.GetDeviceData("Lat"))+"\n")

# barometric pressure and altitude

builder.write("Pressure:" + str(deviceModel.GetDeviceData("Pressure"))+"\t")

builder.write("Height:" + str(deviceModel.GetDeviceData("Height"))+"\n")

# GPS: altitude, heading angle, GPS ground speed

builder.write("GPSHeight:" + str(deviceModel.GetDeviceData("GPSHeight")) + "\t")

builder.write("GPSYaw:" + str(deviceModel.GetDeviceData("GPSYaw"))+"\t")

builder.write("GPSV:" + str(deviceModel.GetDeviceData("GPSV")) + "\n")
```

```
# Four elements
builder.write("Q0:" + str(deviceModel.GetDeviceData("Q0"))+"\t")
builder.write("Q1:" + str(deviceModel.GetDeviceData("Q1"))+"\t")
builder.write("Q2:" + str(deviceModel.GetDeviceData("Q2"))+"\t")
builder.write("Q3:" + str(deviceModel.GetDeviceData("Q3"))+"\n")
# The port number
builder.write("D0:" + str(deviceModel.GetDeviceData("D0"))+"\t")
builder.write("D1:" + str(deviceModel.GetDeviceData("D1"))+"\t")
builder.write("D2:" + str(deviceModel.GetDeviceData("D2"))+"\t")
builder.write("D3:" + str(deviceModel.GetDeviceData("D3"))+"\n")
# Positioning accuracy: number of satellites, position accuracy, horizontal accuracy, vertical accuracy
builder.write("SVNUM:" + str(deviceModel.GetDeviceData("SVNUM")) + "\t")
builder.write("PDOP:" + str(deviceModel.GetDeviceData("PDOP"))+"\t")
```

```
builder.write("HDOP:" + str(deviceModel.GetDeviceData("HDOP")) + "\t")
builder.write("VDOP:" + str(deviceModel.GetDeviceData("VDOP"))+"\n")
# version number
builder.write("VersionNumber:" + str(deviceModel.GetDeviceData("VersionNumber")) + "\n")

print(builder.getvalue())
```

Record data

The data of the sensor can be obtained through the JY901 object, but usually the host computer needs to record the data of the sensor. JY901 has an OnRecord event that will notify you when the data should be recorded, and the OnRecord event can be realized when the device is turned on;) method to record data

Config the sensor

The sensor can be operated by the method of JY901

JY901.UnlockReg() Send unlock register command

JY901.AppliedCalibration() Sends the addition calibration command

JY901.StartFieldCalibration() Send start field calibration command

JY901.EndFieldCalibration() Send end field calibration command

JY901.SendProtocolData() Send other commands

Acceleration calibration

Add-on calibration of the sensor by calling the JY901.AppliedCalibration() method

```
def AppliedCalibration(JY901):
    """
    Addition calibration
    :param JY901: Device model
    :return:
    """
    if JY901.IsOpen():
        # Unlock the register and send the command
        JY901.UnlockReg()
        # Add up calibration
        JY901.AppliedCalibration()

        # The following two lines are equivalent to the above, it is recommended to use the above
```

```
        # Unlock the register and send the command
        # JY901.SendProtocolData([0xff, 0xaa, 0x69, 0x88, 0xb5], 50)
        # Add up calibration
        # JY901.SendProtocolData([0xff, 0xaa, 0x01, 0x01, 0x00], 4000)

        print("Completion of total calibration")
    else:
        print("The device is not open")
```

Magnetic Field Calibration

Perform magnetic field calibration on the sensor by calling the JY901.StartFieldCalibration() method and the JY901.EndFieldCalibration()

method

```
def StartFieldCalibration(JY901):
    """
    Magnetic Field Calibration
    :param JY901: Device model
    :return:
    """
    if JY901.IsOpen():
        # Unlock the register and send the command
        JY901.UnlockReg()
        # start magnetic field calibration
        JY901.StartFieldCalibration()
```

```
        # The following two lines are equivalent to the above, it is recommended to use the above
        # Unlock the register and send the command
        # JY901.SendProtocolData([0xff, 0xaa, 0x69, 0x88, 0xb5], 50)
        # start magnetic field calibration
        # JY901.SendProtocolData([0xff, 0xaa, 0x01, 0x07, 0x00], 100)

        if input("Please make a slow rotation around the XYZ axis respectively, after the three-axis rotation is completed, end the calibration
(Y/N)?").lower() == "y":
            # Unlock the register and send the command
            JY901.UnlockReg()
            # end magnetic field calibration
            JY901.EndFieldCalibration()
```

```
            # The following two lines are equivalent to the above, it is recommended to use the above
            # Unlock the register and send the command
            # JY901.SendProtocolData([0xff, 0xaa, 0x69, 0x88, 0xb5], 50)
            # start magnetic field calibration
            # JY901.SendProtocolData([0xff, 0xaa, 0x01, 0x00, 0x00], 100)

            print("End magnetic field calibration")
    else:
        print("The device is not open")
```

More

For other operations, please refer to the sensor manual

Read sensor register

The sensor register can be read through the JY901.SendReadReg() method, or the JY901.SendProtocolData() method can be used

After sending the read command, the register value will be saved in JY901, and the register data needs to be obtained through JY901.GetDeviceData().

```
def IsReadReg(JY901, reg, waitTime):
    """
    read register
    :param JY901: Device model
    :param reg: register address
    :param waitTime: wait time
    :return:
```

```
"""
bRet = False
if JY901.IsOpen():
    # read register
    if JY901.SendReadReg(reg, waitTime):
        bRet = True
    else:
        print(str(reg) + "Failed to read")
else:
    print("The device is not open")

return bRet
```

JY901 API

Method	Directions	Parameter introduction	Return value
void SetPortName(string portName)	Set the serial port to be opened	portName: serial port number	void
void SetBaudrate(int baudRate)	Specifies the baud rate to be turned on	baudRate: baud rate	void
void Open() Open	Turn on the device	NO	void

bool IsOpen()	Whether the device is open	NO	Return whether to open open: true off: false
void Close()	Turn off the device	NO	void
void SendProtocolData(byte[] data, int waitTime)	Send the data with the protocol and specify the waiting time	data: data to be sent waitTime: wait time	void

void SendReadReg(byte reg, int waitTime)	Send the command to read the register	reg: command to be sent wait time: wait time	void
void UnlockReg()	unlock register	NO	void
void SaveReg()	save register	N0	void
void AppliedCalibration()	Acceleration calibration	NO	void
void StartFieldCalibration()	Start magnetic	NO	void

	field calibration		
void EndFieldCalibration()	end magnetic field	NO	void
void SetReturnRate(byte rate)	Set return rate	rate: the return rate to be set	void
string GetDeviceName()	get device name	NO	return device name
string GetDeviceData(string key)	Get key value	key: data key value	return data

	data		value