Instruction Manual

from Modu-Link

Remote Access Controller

A computer on a desk

Description automatically generated

Updated: December 2, 2024

This document was created by Modu-Link. All ideas and content are the property of Modu-Link. You are welcome to use our product and our base code for your own personal or academic projects if all Modu-Link work is properly cited, recognized, and given credit.

**About:**

The remote access controller (RAC) is a low-cost, open-source device that is designed to enable remote access to connected devices and sensors. Utilizing local Wi-Fi networks, the RAC empowers a wide variety of potential internet of things (IOT) applications.

This RAC is intended to provide a flexible and scalable utility for hobbyists and professionals alike. Our goal with this design is to provide a foundation for the end user to further develop and adapt to their use case. With some general knowledge of programming and microcontrollers, users can adapt RAC to an array of use cases, from smart home systems to industrial monitoring.

**Overview Diagram:**

A diagram of a diagram

Description automatically generated with medium confidence

User: The user is anyone interacting with the controlling device to change the behaviour of the remote controller. The user does this by interacting with the graphical user interface (GUI) on the controlling device.

GUI: The GUI allows the user to control the data logging of the sensor input and to plot graphs on the controlling device for the user to see. The GUI also sends commands to the processor to send to the remote device and it receives the data from the processor to plot.

Processor: The processor handles the requests from the GUI and sends communication back and forth from the remote device via the wireless transmitter and receiver.

Wireless Transmitter and Receiver: The wireless transmitter and receiver on the controlling device communicate with the wireless receiver and transmitter of the remote device via Wi-Fi.

Wireless Transmitter and Receiver (Remote): The wireless transmitter and receiver on the remote device communicate with the wireless receiver and transmitter of the controller device via Wi-Fi. They also interact with the processor on the remote device to send commands received from the controlling device and to send data from the input and output (I/O) of the remote device.

Processor (Remote): The processor on the remote device receives commands from the controlling device and either returns data from the sensors or not, depending on the commands. The processor interacts with the I/O to sample the sensor data.

I/O: The I/O is connected to the remote sensors and acts as a mechanical interface between the processor and the sensors.

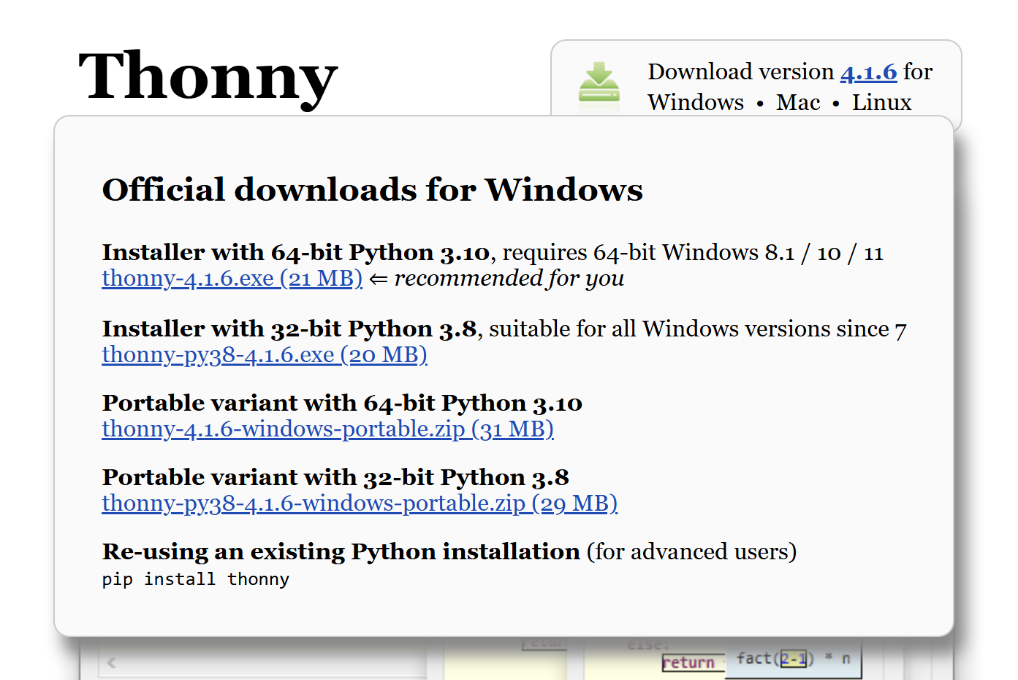
Sensors: The sensors can be a variety of sensors, so long as the power requirements of the setup are required and that there are enough pins remaining on the remote device to add all the desired sensors.

**Requirements:**

A controlling device that has wireless communication available is required. It should have the Thonny IDE installed and the ability to run a python script. A network which has a known SSID and password is also required.

**Software installation:**

Here are the steps required to install the Thonny IDE. Other IDEs may also be supported for the ESP32 if you do not want to use Thonny.



(1)

(2)

Hover over the Windows link (1) and then click the appropriate Thonny application file (2).

Download and launch the installer.

A screenshot of a computer

Description automatically generated

Follow the prompts to install Thonny. Use the default settings unless you have a preference and chose to change the settings.

A screenshot of a computer

Description automatically generated

You should get this message when it is done! Click Finish.



Make sure the settings are your desired language and that the initial settings are set to Standard. Then click Let’s go!

**Sample project:**

The following are two codes you will need for the project. The first one is to be uploaded to the remote device via the Thonny IDE and the other file can be executed with any python script

The first thing you will need to know is the SSID and the login of your network. Enter those into the first file (in Thonny) and then upload your code to the ESP32 (don’t forget to plug it in and select the right com port if required). The output should look as follows:

Connecting… connecting… connecting…

Then IP address

Then run the second file and type in the IP address of the ESP32 into the web browser to connect. Now have fun controlling the remote access controller. You can use a TMP36 temperature sensor with the output connected to pin 39 of the ESP32 for a quick test project.

To stop the program, type CRTL+C. Anywhere into Thonny Ctrl + C kills program

**File 1:**

import socket  
import network  
import time  
import struct  
from machine import ADC, Pin, reset

# Wi-Fi credentials  
ssid = 'ORBI90'  
password = 'PICODEMO'

# Server Configuration  
SERVER\_IP = '192.168.1.5'  # Server IP address  
PORT = 12345  # Port number

def connect\_wifi():  
    wlan = network.WLAN(network.STA\_IF)  
    wlan.active(True)  
    wlan.connect(ssid, password)  
    print("Connecting to Wi-Fi...")  
    timeout = 10  
    while not wlan.isconnected() and timeout > 0:  
        time.sleep(1)  
        timeout -= 1  
    if wlan.isconnected():  
        print(f"Connected to Wi-Fi. IP: {wlan.ifconfig()[0]}")  
        return wlan.ifconfig()[0]  
    else:  
        print("Failed to connect to Wi-Fi.")  
        return None

def send\_receive\_data():  
    Polling = 0  # Default polling state (off)  
    rate = 0.1  # Set sample rate to 0.1 seconds  
    while True:  
        try:  
            s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
            s.connect((SERVER\_IP, PORT))  
            s.setblocking(False)  # Set socket to non-blocking mode  
            print(f"Connected to server at {SERVER\_IP}:{PORT}")  
            device\_id = 1  # Unique device ID for the client

            # Receive initial command from server  
            while True:  
                try:  
                    command = s.recv(1024).decode("utf-8")  
                    print(f"Received from server: {command}")  
                    if command == "START":  
                        print("Starting data transmission...")  
                        Polling = 1  
                        break  
                except:  
                    pass  # No command received yet

            while True:  
                if Polling == 1:  
                    # Read temperature from external sensor  
                    adc1 = ADC(39)  
                    tempsensevoltage = adc1.read\_uv() / 1000  
                    Temp = (tempsensevoltage - 500) / 10  # Temperature in Celsius

                    # Read light intensity from photodiode  
                    adc2 = ADC(Pin(34))  
                    photodiodevoltage = adc2.read\_uv() / 1000  # Convert to mV

                    # Prepare data transfer  
                    timestamp = time.time()  
                    data = [Temp, photodiodevoltage]

                    # Send the number of variables  
                    num\_variables = len(data)  
                    header = struct.pack('I', num\_variables)  
                    s.send(header)

                    # Send the device ID  
                    device\_id\_header = struct.pack('I', device\_id)  
                    s.send(device\_id\_header)

                    # Send the timestamp  
                    timestamp\_header = struct.pack('d', timestamp)  
                    s.send(timestamp\_header)

                    # Send the payload  
                    payload = struct.pack(f"{'f' \* num\_variables}", \*data)  
                    s.send(payload)

                    # Try to receive any commands from the server  
                    try:  
                        command = s.recv(1024).decode("utf-8")  
                        print(f"Received from server: {command}")  
                        if command == "STOP":  
                            print("Stopping data transmission...")  
                            Polling = 0  
                        elif command.startswith("RATE = "):  
                            try:  
                                rate = float(command.split("=")[1].strip())  
                                print(f"Setting sample rate to {rate} seconds.")  
                            except ValueError:  
                                print("Invalid sample rate value.")  
                        elif command == "RESET":  
                            print("Resetting ESP32...")  
                            reset()  
                    except:  
                        pass  # No command received

                    time.sleep(rate)  
                else:  
                    # Try to receive any commands from the server  
                    try:  
                        command = s.recv(1024).decode("utf-8")  
                        print(f"Received from server: {command}")  
                        if command == "START":  
                            print("Starting data transmission...")  
                            Polling = 1  
                        elif command == "RESET":  
                            print("Resetting ESP32...")  
                            reset()  
                    except:  
                        pass  # No command received  
                    time.sleep(0.1)  # Sleep briefly to prevent tight loop

        except (OSError, Exception) as e:  
            print(f"Error: {e}")  
            s.close()  
            print("Attempting to reconnect in 2 seconds...")  
            time.sleep(2)  
            reset()

if \_\_name\_\_ == '\_\_main\_\_':  
    wifi\_ip = connect\_wifi()  
    if wifi\_ip is None:  
        print("Exiting program due to Wi-Fi connection failure.")  
        reset()  
    send\_receive\_data()

**File 2 (Server, laptop/phone/computer side):**

import socket

import struct

import matplotlib.pyplot as plt

from matplotlib.animation import FuncAnimation

from matplotlib.widgets import Button

from threading import Thread

import queue

HOST = '0.0.0.0'  # Listen on all interfaces

PORT = 12345      # Port number (ensure it's the same on both ends)

temperature\_data = queue.Queue()

light\_intensity\_data = queue.Queue()

def recvall(sock, n):

    """Helper function to receive n bytes or return None if EOF is hit."""

    data = b''

    while len(data) < n:

        packet = sock.recv(n - len(data))

        if not packet:

            return None

        data += packet

    return data

def server\_thread():

    server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    server\_socket.bind((HOST, PORT))

    server\_socket.listen(1)

    print(f"Server listening on {HOST}:{PORT}...")

    try:

        while True:

            client\_socket, client\_address = server\_socket.accept()

            print(f"Connection established with {client\_address}")

            # Send "START" command to client

            command = "START"

            client\_socket.send(command.encode())

            while True:

                # Receive the number of variables (4 bytes)

                data = recvall(client\_socket, 4)

                if not data:

                    break

                num\_variables = struct.unpack('I', data)[0]

                # Receive the device ID (4 bytes)

                data = recvall(client\_socket, 4)

                if not data:

                    break

                device\_id = struct.unpack('I', data)[0]

                # Receive the timestamp (8 bytes)

                data = recvall(client\_socket, 8)

                if not data:

                    break

                timestamp = struct.unpack('d', data)[0]

                # Receive the payload (num\_variables \* 4 bytes)

                data = recvall(client\_socket, num\_variables \* 4)

                if not data:

                    break

                values = struct.unpack(f"{'f' \* num\_variables}", data)

                print(f"Received from device {device\_id} at {timestamp}: {values}")

                # Put data into queues

                temperature\_data.put(values[0])

                light\_intensity\_data.put(values[1])

                # Send acknowledgment

                response = "Data received"

                client\_socket.send(response.encode())

            client\_socket.close()

            print(f"Connection closed with {client\_address}")

    except KeyboardInterrupt:

        print("Shutting down server.")

    finally:

        server\_socket.close()

def update\_plot(frame):

    if plotting and not temperature\_data.empty() and not light\_intensity\_data.empty():

        temp = temperature\_data.get()

        light = light\_intensity\_data.get()

        x\_data.append(len(x\_data))  # Time or sequence index

        temp\_data.append(temp)

        light\_data.append(light)

        temp\_line.set\_data(x\_data, temp\_data)

        ax1.set\_xlim(max(0, len(x\_data) - 100), len(x\_data))  # Keep only the last 100 points

        ax1.set\_ylim(min(temp\_data[-100:]), max(temp\_data[-100:]))

        light\_line.set\_data(x\_data, light\_data)

        ax2.set\_xlim(max(0, len(x\_data) - 100), len(x\_data))  # Keep only the last 100 points

        ax2.set\_ylim(min(light\_data[-100:]), max(light\_data[-100:]))

server = Thread(target=server\_thread, daemon=True)

server.start()

x\_data = []        # X-axis (time or sequence index)

temp\_data = []     # Y-axis for temperature

light\_data = []    # Y-axis for light intensity

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))

temp\_line, = ax1.plot([], [], label='Temperature (°C)')

ax1.set\_title("Temperature Data")

ax1.set\_xlabel("Time")

ax1.set\_ylabel("Temperature (°C)")

ax1.legend()

ax1.grid(True)

light\_line, = ax2.plot([], [], label='Light Intensity (V)', color='orange')

ax2.set\_title("Light Intensity Data")

ax2.set\_xlabel("Time")

ax2.set\_ylabel("Light Intensity (V)")

ax2.legend()

ax2.grid(True)

plt.tight\_layout()

# Add buttons

from matplotlib.widgets import Button

# Adjust the main plot to make room for buttons

plt.subplots\_adjust(bottom=0.2)

ax\_start = plt.axes([0.7, 0.05, 0.1, 0.075])

ax\_stop = plt.axes([0.81, 0.05, 0.1, 0.075])

btn\_start = Button(ax\_start, 'Start')

btn\_stop = Button(ax\_stop, 'Stop')

plotting = False

def start\_plotting(event):

    global plotting

    plotting = True

    # Clear data when starting

    x\_data.clear()

    temp\_data.clear()

    light\_data.clear()

    # Clear queues

    while not temperature\_data.empty():

        temperature\_data.get()

    while not light\_intensity\_data.empty():

        light\_intensity\_data.get()

    print("Plotting started.")

def stop\_plotting(event):

    global plotting

    plotting = False

    print("Plotting stopped.")

btn\_start.on\_clicked(start\_plotting)

btn\_stop.on\_clicked(stop\_plotting)

ani = FuncAnimation(fig, update\_plot, interval=20)  # Update every 100ms

plt.show()

A circuit board with wires

Description automatically generated

Enjoy your own personal remote access controller!

For any questions or technical assistance, please call the number provided by your local distributor.