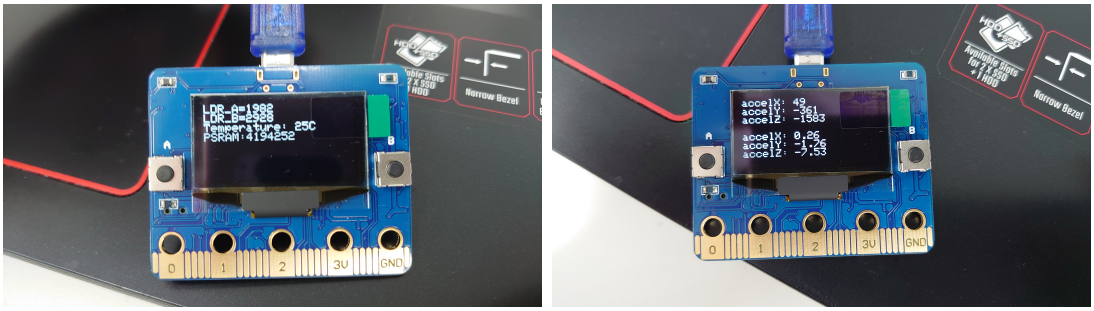
物聯網實務第十周作業

電機四乙10828241 陳大荃

November 16, 2022

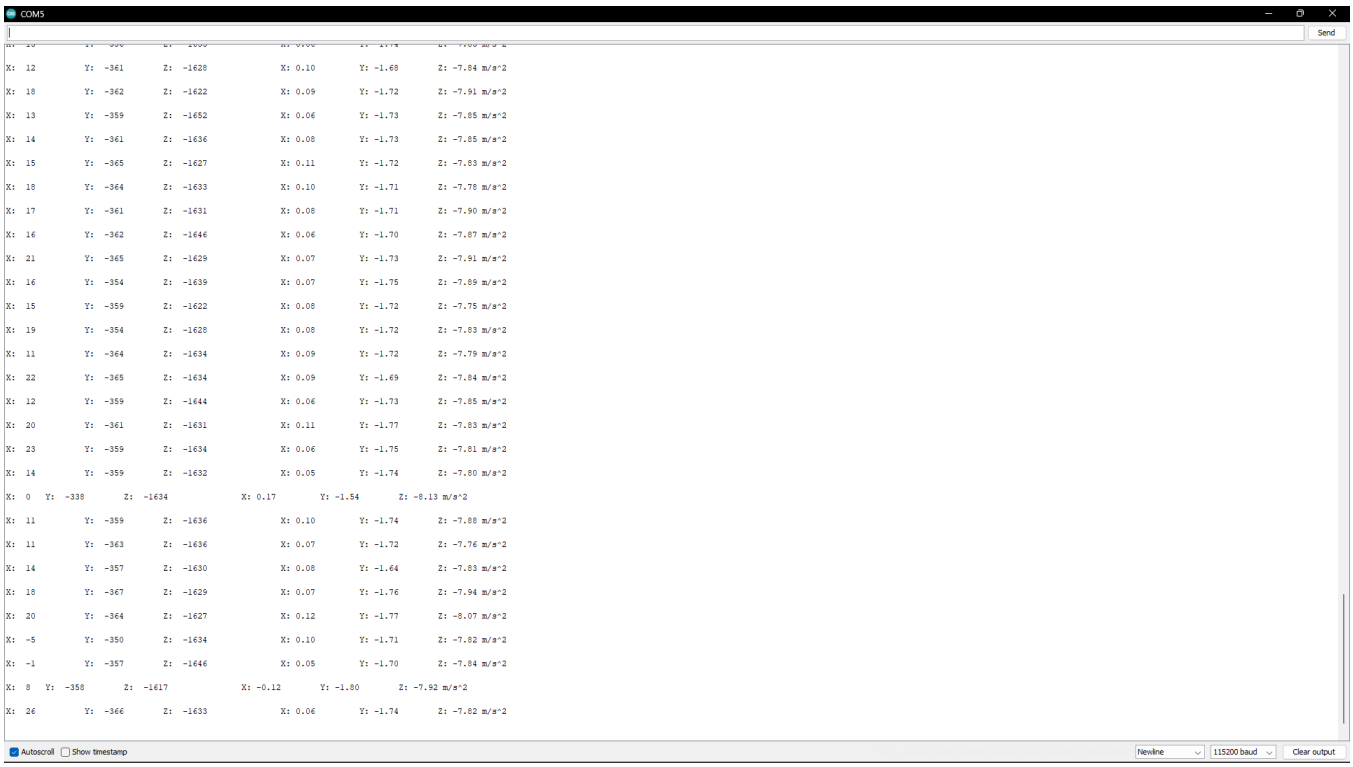
Exercise 9-1 Test pocketcard_demo.



(a) Button 1 (b) Button 2

Figure 1: Exercise 9-1.

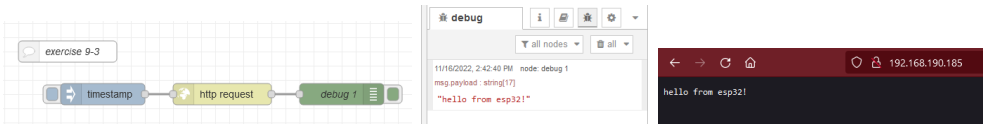
Exercise 9-2 Test MSA301_Accelerometer_Demo.



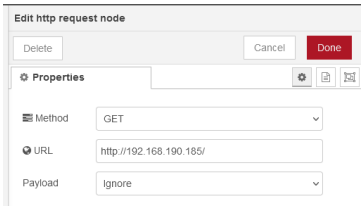
(a) axis accelerometer values

Figure 2: Exercise 9-2.

Exercise 9-3 Hello from ESP32.



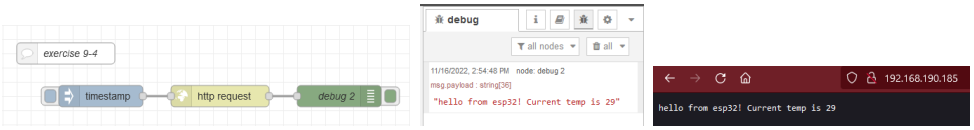
(a) flow (b) node: debug (c) node: webpage



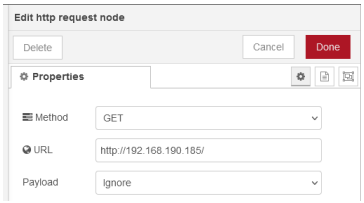
(d) node: http request

Figure 3: Exercise 9-3.

Exercise 9-4 Get current temp from the web server I.



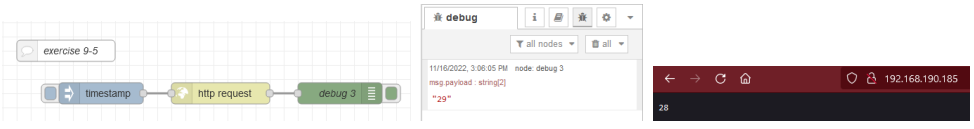
(a) flow (b) node: debug (c) node: webpage



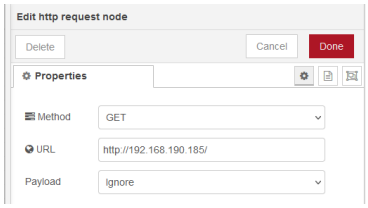
(d) node: http request

Figure 4: Exercise 9-4.

Exercise 9-5 Get current temp from the web server II.



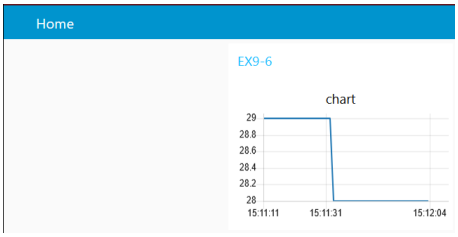
(a) flow (b) node: debug (c) node: webpage



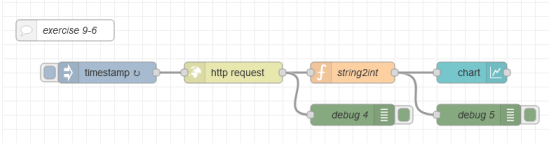
(d) node: http request

Figure 5: Exercise 9-5.

Exercise 9-6 Get current temp from the web server III.



(a) node: dashboard



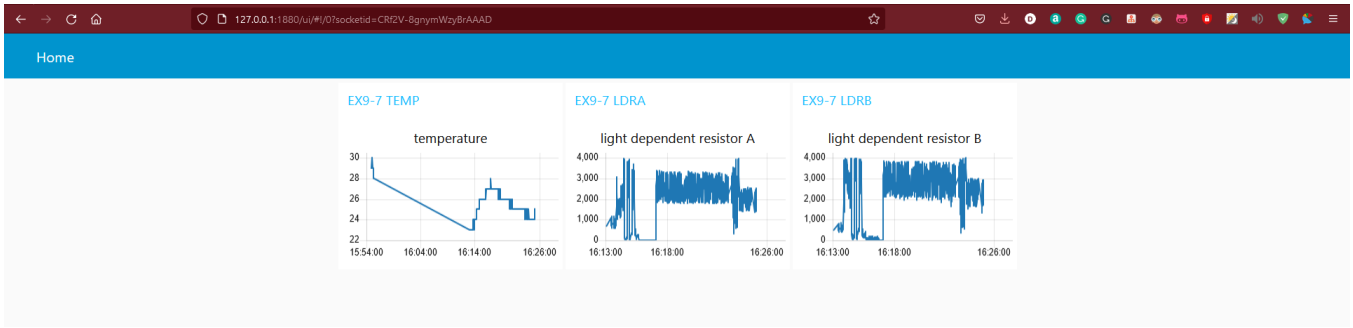
(b) flow

(c) node: http request

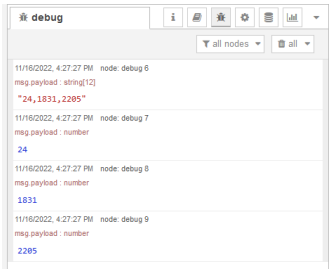
(d) node: string2int

Figure 6: Exercise 9-6.

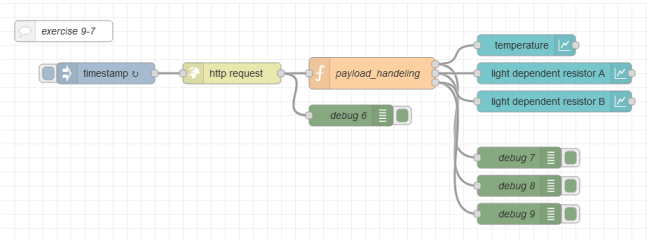
Exercise 9-7 Show another sensor values from the esp32 board on the Node-RED dashboard.



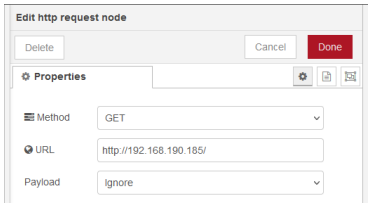
(a) node: dashboard



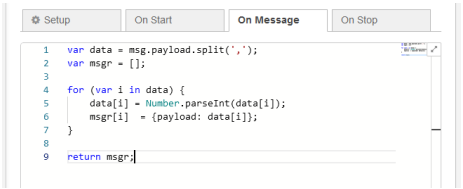
(b) node: debug



(c) flow



(d) node: http request



(e) node: payload handeling

Figure 7: Exercise 9-7.

C code for ESP32 dev board, Exercise 9-7.

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include <ESPmDNS.h>

#define LDRA_IO 39
#define LDRB_IO 36
#define THERMISTOR_IO 34

// Series resistor value
#define SERIESRESISTOR 10000
// Number of samples to average
#define SAMPLERATE 5
// Nominal resistance at 25C
#define THERMISTORNOMINAL 10000
// Nominal temperature in degrees
#define TEMPERATURENOMINAL 25
// Beta coefficient
#define BCOEFFICIENT 3380

const char* ssid = ".........";
const char* password = "..........";

WebServer server(80);

const int led = 13;

int getTemp() {
    double thermalSamples[SAMPLERATE];
    double average, kelvin, resistance, celsius;
    int i;
    // Collect SAMPLERATE (default 5) samples
    for (i=0; i<SAMPLERATE; i++) {
        thermalSamples[i] = analogRead(THERMISTOR_IO);
        delay(10);
    }
    // Calculate the average value of the samples
    average = 0;
    for (i=0; i<SAMPLERATE; i++) {
        average += thermalSamples[i];
    }
    average /= SAMPLERATE;
    // Convert to resistance
    resistance = 4095 / average - 1;
    resistance = SERIESRESISTOR / resistance;
    /*
     * Use Steinhart equation (simplified B parameter equation) to convert resistance to kelvin
     * B param eq: T = 1/( 1/To + 1/B * ln(R/Ro) )
     * T  = Temperature in Kelvin
     * R  = Resistance measured
     * Ro = Resistance at nominal temperature
     * B  = Coefficent of the thermistor
     * To = Nominal temperature in kelvin
     */
    //kelvin = resistance/THERMISTORNOMINAL;                                  // R/Ro
    kelvin = THERMISTORNOMINAL/resistance;                                  // R/Ro
    kelvin = log(kelvin);                                                                   // ln(R/Ro)
    kelvin = (1.0/BCOEFFICIENT) * kelvin;                                   // 1/B * ln(R/Ro)
    kelvin = (1.0/(TEMPERATURENOMINAL+273.15)) + kelvin;  // 1/To + 1/B * ln(R/Ro)
    kelvin = 1.0/kelvin;
    // Convert Kelvin to Celsius
    celsius = kelvin - 273.15;
    // Send the value back to be displayed
    return celsius;
}

int getLDRA() {return analogRead(LDRA_IO);}
int getLDRB() {return analogRead(LDRB_IO);}

void handleRoot() {
  digitalWrite(led, 1);
  int temp = getTemp();
  int ldra = getLDRA();
  int ldrb = getLDRB();
//  int msax = getMSAX();
//  int msay = getMSAY();
//  int msaz = getMSAZ();
//  Serial.println(String(msax) + "/" + String(msay) + "/" + String(msaz));
  server.send(200, "text/plain", String(temp) + "," + String(ldra) + "," + String(ldrb));
  digitalWrite(led, 0);
}

void handleNotFound() {
  digitalWrite(led, 1);
  String message = "File Not Found\n\n";
  message += "URI: ";
  message += server.uri();
  message += "\nMethod: ";
  message += (server.method() == HTTP_GET) ? "GET" : "POST";
  message += "\nArguments: ";
  message += server.args();
  message += "\n";
  for (uint8_t i = 0; i < server.args(); i++) {
    message += " " + server.argName(i) + ": " + server.arg(i) + "\n";
  }
  server.send(404, "text/plain", message);
  digitalWrite(led, 0);
}

void setup(void) {
  pinMode(led, OUTPUT);
  digitalWrite(led, 0);
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
```

```
  WiFi.begin(ssid, password);
  Serial.println("");

  // Wait for connection
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  if (MDNS.begin("esp32")) {
    Serial.println("MDNS responder started");
  }

  server.on("/", handleRoot);

  server.on("/inline", []() {
    server.send(200, "text/plain", "this works as well");
  });

  server.onNotFound(handleNotFound);

  server.begin();
  Serial.println("HTTP server started");
}

void loop(void) {
  server.handleClient();
}
```