Relazione progetto
Prototipizzazione con arduino

Venturi Davide VR471414
Vilotto Tommaso VR471487

Indice

1 Introduzione **3**

2 Tabella dei Componenti **4**

3 Realizzazione **5**

4 Descrizione Codice e Librerie **6**

5 Schema Collegamenti **10**

6 Implementazione Fisica **11**

7 Riferimenti **12**

1 Introduzione

La nostra idea era quella di creare un regolatore di temperatura che ci permettesse di mantenere temperature stabili entro piccoli range, adatto quindi più ad un uso industriale piuttosto che casalingo.

Il termostato rileva con una frequenza (aggiustabile) di ogni secondo la temperatura tramite una termocoppia, per una migliore precisione, e in base ai valori registrati e alla temperatura impostata decide se vi è necessità di scaldare o raffreddare l'ambiente. Tramite opportuna logica evitiamo che vi sia un susseguirsi di on/off e lavoriamo su un determinato range per cercare di fare salvaguarda di risorse.
L'interfacciamento col termostato avviene tramite un piccolo schermo oled che presenta 3 schermate:

- **Main Page**: dove vediamo la temperatura rilevata, la temperatura goal e se il nostro termostato ha attivato riscaldamento, raffreddamento;
- **Second Page**: dove possiamo impostare la temperatura goal che vogliamo raggiungere;
- **Off Page**: se il termostato viene spento andremo in questa pagina dove verrà semplicemente visualizzata la temperatura corrente.

Inoltre, il termostato invierà periodicamente i dati rilevati ed elaborati in cloud e sarà quindi possibile accedere da remoto per visualizzarne l'andamento temporale.

Per rendere possibile il tutto abbiamo pensato ad un architettura master/slave di tipo seriale rx/tx nella quale il nostro arduino uno (slave) esegue le rilevazioni, gestisce il monitor, esegue le computazioni e invia i dati all'esp32 (master) che si occupa di mandare il tutto in cloud.

Come piattaforma cloud ci siamo appoggiati a Thingspeak, il quale offre di base una visualizzazione dei dati ricevuti e inoltre permette la creazione di propri grazie a matlab.

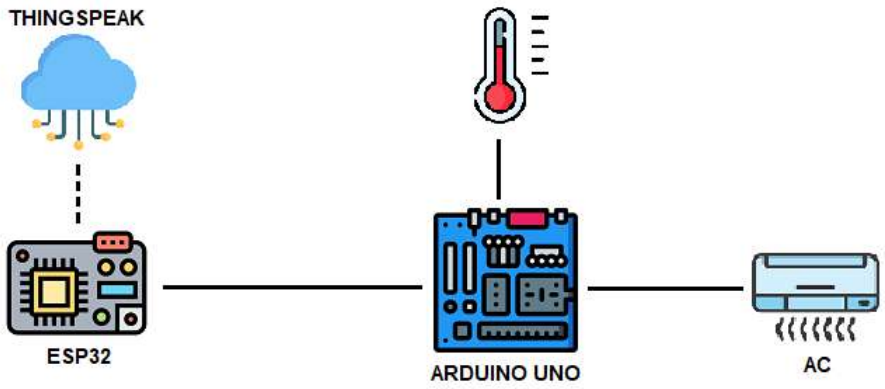


Immagine 1: Schema del progetto ad alto livello

2 Tabella dei Componenti

Componenti	
Lato Arduino	Lato Esp32
Arduino Uno Schermo oled Potenziometro Bottone Resistenza (1k ohm) Switch Led Relé (idealmente collegati ad un ac) Termocoppia	Esp32

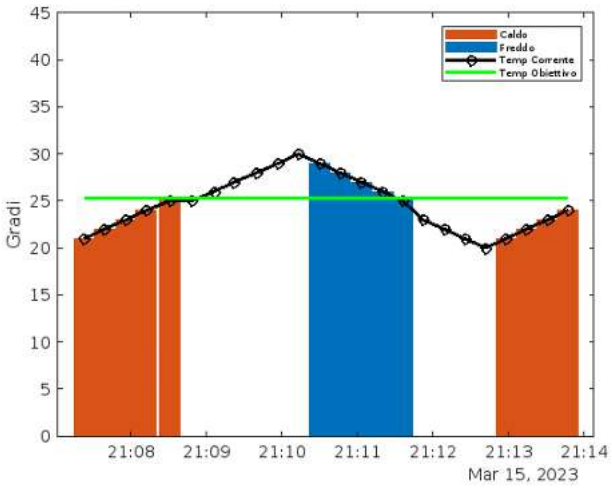
3 Realizzazione

Per la realizzazione prima di tutto ci siamo avvalorati dell'uso di un sito per simulare i componenti online: wokwi. Nell'attesa di ricevere fisicamente i componenti abbiamo appunto completato lo sketch di arduino e dell'esp simulando il passaggio di dati tra i due microcontrollori, che online non si poteva fare. Abbiamo scritto buona parte del codice sviluppando contemporaneamente lo schema fisico dei componenti su Fritzing.

E' stata nostra premura focalizzarci attentamente sul controllo del raggiungimento della temperatura obiettivo. Volendo abbracciare la filosofia PID abbiamo realizzato un algoritmo per gestire al meglio lo spegnimento/accensione di pompa di calore e raffreddamento in maniera intelligente. Tramite il potenziometro si imposta un obiettivo di calore (goal) da raggiungere, arduino elaborerà un intervallo di "accettabilità" pari a ± 0.5gradi entro i quali viene accettata la temperatura rilevata. Se però era in atto un riscaldamento/raffreddamento, quest'ultimo non viene spento prima di aver raggiunto una soglia maggiore uguale a +0.25 al goal se stava riscaldando, o minore uguale a -0.25 al goal se stava raffreddando.

Una volta ricevuti i componenti abbiamo iniziato l'assemblaggio vero e proprio del progetto, rinfacciando subito dei problemi con certi componenti. Abbiamo sostituito lo schermo oled e il potenziometro, il cui feedback non rispettava il comportamento che ci serviva. Dopo aver montato tutto come da schema siamo passati alla manipolazione dei dati in cloud.

Per il cloud abbiamo scelto Thingspeak in quanto nella sua licenza gratis riusciva ad adempiere a tutte le nostre richieste. Ogni 16 secondi salviamo dei dati in cloud in 4 canali differenti: temperatura attuale, temperatura obiettivo, bit di flag riscaldamento attivo, bit di flag raffreddamento attivo. Successivamente abbiamo unito i 4 canali dati in uno solo, usando mathlab siamo arrivati ad avere un unico grafico per visualizzare in chiaro tutti i dati.



Infine, abbiamo creato una pagina php responsive, hostata su Altervista, per visualizzare comodamente da pc o smartphone i dati elaborati da mathlab. Il frame della finestra di mathlab purtroppo si aggiorna solo ogni 15 secondi, causa limiti della licenza, la temperatura attuale invece ogni 16.

Link al sito: `http://checkitout.altervista.org/dati.php`

4 Descrizione Codice e Librerie

Codice Arduino:

```
void setup() {

  pinMode ( 2 , INPUT_PULLUP ); //thermo mode on/off
  pinMode ( 4 , INPUT_PULLUP ); //swith between thermo on pages
  pinMode ( A4 , OUTPUT ); //relay output for cooling
  pinMode ( A5 , OUTPUT ); //relay output for heating
  Serial.begin(9600);
  u8g.setColorIndex(1);          // set color to white

  for (int i = 0; i < 180; i++){    // pre-calculate x and y positions into the look-up tables
     precalculated_x_radius_pixel[i] =  sin(radians(i-90)) * radius_pixel + center_x;
     precalculated_y_radius_pixel[i] = -cos(radians(i-90)) * radius_pixel + center_y;
  }

  thermo.begin(MAX31865_3WIRE);  // set to 2WIRE or 4WIRE as necessary

}
```

Immagine 2:
Inizialmente settiamo i pin:

- Switch collegatto a pin 2;
- Bottone collegato a pin 4;
- A4 e A5 collegati ad i relé.

Successivamente precalcoliamo i gradi per la pagina di selezione della temperatura, questo procedimento alleggerisce il caricamento della schermata.

```
void loop() {
  readButtons();
  if(buttonStatus1 == false)
   firstRead = true;
  if ( buttonStatus == true || (buttonStatus1 == false && buttonStatus == false)){
    mainPage();
  }
  else{
    secondPage();
  }
  sendData();
}
```

Immagine 3: In loop, in base a quali flag sono segnati, decidiamo quale pagina mostrare a schermo.

```
void mainPage(void){
  if(true){
    unsigned long now = millis();
    if(lastSampleTime > now){
      lastSampleTime = 0;
      firstRead = true;
    }
    if (now - lastSampleTime >= deltaTime || firstRead){
      lastSampleTime += deltaTime;
      uint16_t rtd = thermo.readRTD();
      temp = thermo.temperature(RNOMINAL, RREF);
      if(firstRead){
         firstRead = false;
      }
      if((temp>=(potentiometer_value/10.0)-RANGE && temp<=(potentiometer_value/10.0)+RANGE) || !buttonStatus1){
        bloccaAC = true;
      }
      else{
        bloccaAC = false;
      }
      if(ac && buttonStatus1){
        if((temp>=(potentiometer_value/10.0)+0.25 && hot == true)  || (temp<=(potentiometer_value/10.0)-0.25 && cold == true)){
          digitalWrite(FREDDO, LOW);
          digitalWrite(CALDO, LOW);
          ac = false;
          hot = false;
          cold = false;
        }
        else
          tempHandler();
      }
      else if(!bloccaAC){
        tempHandler();
      }
    }
  }
```

Immagine 4: Qui vediamo la schermata principale, ogni secondo leggiamo la temperatura e a seconda del valore letto decidiamo se spegnere il riscaldamento/raffreddamento oppure chiamiamo la funzione sottostante.

```
void tempHandler(void){
  if(temp>(potentiometer_value/10.0)+RANGE){
        digitalWrite(FREDDO, HIGH);
        digitalWrite(CALDO, LOW);
        ac = true;
        cold = true;
        hot = false;
      }
  else if(temp<(potentiometer_value/10.0)-RANGE){
        digitalWrite(FREDDO, LOW);
        digitalWrite(CALDO, HIGH);
        ac = true;
        hot = true;
        cold = false;
    }
}
```

Immagine 5: La funzione tempHandler viene eseguita per verificare se ci troviamo fuori dal range ideale e quindi accendere il riscaldamento/raffreddamento.

```
void sendData(void){
  unsigned long now = millis();
  if(lastSampleTime_2 > now){   //if millis resetted (c.a. 5 days)
    lastSampleTime_2 = 0;
  }
  if (now - lastSampleTime_2 >= deltaTime_2){ //it reads the temperature after first boot and then after every 5 minutes
      lastSampleTime_2 += deltaTime_2;

    String apiKey = "EKL504G6U3M10XJK";
    String postStr = apiKey;
    postStr +="&field5=";  //actual temperature
    postStr += String(temp);
    postStr += "&field2="; //temperatura goal
    dtostrf(potentiometer_value/10.0, 1, 2, buffer);
    postStr += buffer;
    postStr += "&field3="; //heating temperature flag
    if(hot)
      postStr += String(1);
    else
      postStr += String(0);
    postStr += "&field4="; //cooling temperature flag
    if(cold)
      postStr += String(1);
    else
      postStr += String(0);
    postStr += "\r\n\r\n\r\n\r\n";
    Serial.print(postStr); //sending string to esp via rx/tx
  }

}
```

Immagine 6: La funzione SendData si occupa della preparazione della stringa di dati da inviare successivamente all'esp32, il quale la invierà in cloud.

```
void readButtons(void){
  tasto = digitalRead(4);
  if(!prev_tasto && tasto)
    prev_tasto = tasto;
  if(prev_tasto && !tasto){
    prev_tasto = tasto;
    buttonStatus = !buttonStatus;
  }
  if(digitalRead ( 2 ) == 1){
    if(buttonStatus1 == false){
      firstRead = true;
      hot = false;
      cold = false;
    }
    buttonStatus1 = true;
  }
  else{
    buttonStatus1 = false;
    cold = false;
    hot = false;
    digitalWrite(FREDDO, LOW);
    digitalWrite(CALDO, LOW);
  }
}
```

Immagine 7: Invocata a ogni loop, questa funzione legge lo stato del bottone, gestendone anche il bouncing, e dello switch.

Codice Esp32:

```
String apiKey = "EKL504G6U3M10XJK";  //API key of the data channel thingspeak
const char* server = "api.thingspeak.com";

void setup() {
  Serial.begin(115200);
  Serial2.begin(9600, SERIAL_8N1, RXp2, TXp2);
  Serial.println("Hello, ESP32!");
  WiFi.begin("quelTom", "password", 6);
  while (WiFi.status() != WL_CONNECTED) {
    delay(100);
    Serial.print(".");
  }
  Serial.println(" Connected!");

}
```

Immagine 8: Inizializzata libreria per il collegamento wifi e comunicazione rx/tx.

```
void loop() {
  if (client.connect(server,80)){

    String data = Serial2.readString();
    client.print("POST /update HTTP/1.1\n");
    client.print("Host: api.thingspeak.com\n");
    client.print("Connection: close\n");
    client.print("X-THINGSPEAKAPIKEY: "+apiKey+"\n");
    client.print("Content-Type: application/x-www-form-urlencoded\n");
    client.print("Content-Length: ");
    client.print(data.length());
    client.print("\n\n");

    client.print(data);
    Serial.print(data);
    Serial.println("Sent data...");
  }
  delay(16000); //lets pause 'cause of the license limit of thingspeak
}
```

Immagine 9: Lettura ogni 16 secondi da seriale dei dati e invio in cloud.

Librerie utilizzate:

- **DallasTemperature** : per la lettura della temperatura
- **U8g** : per la gestione dello schermo oled
- **Wifi** : per connettere l'esp32 a internet
- **OneWire** : per comunicare con i componenti (legato a DallasTemperature)

5 Schema Collegamenti

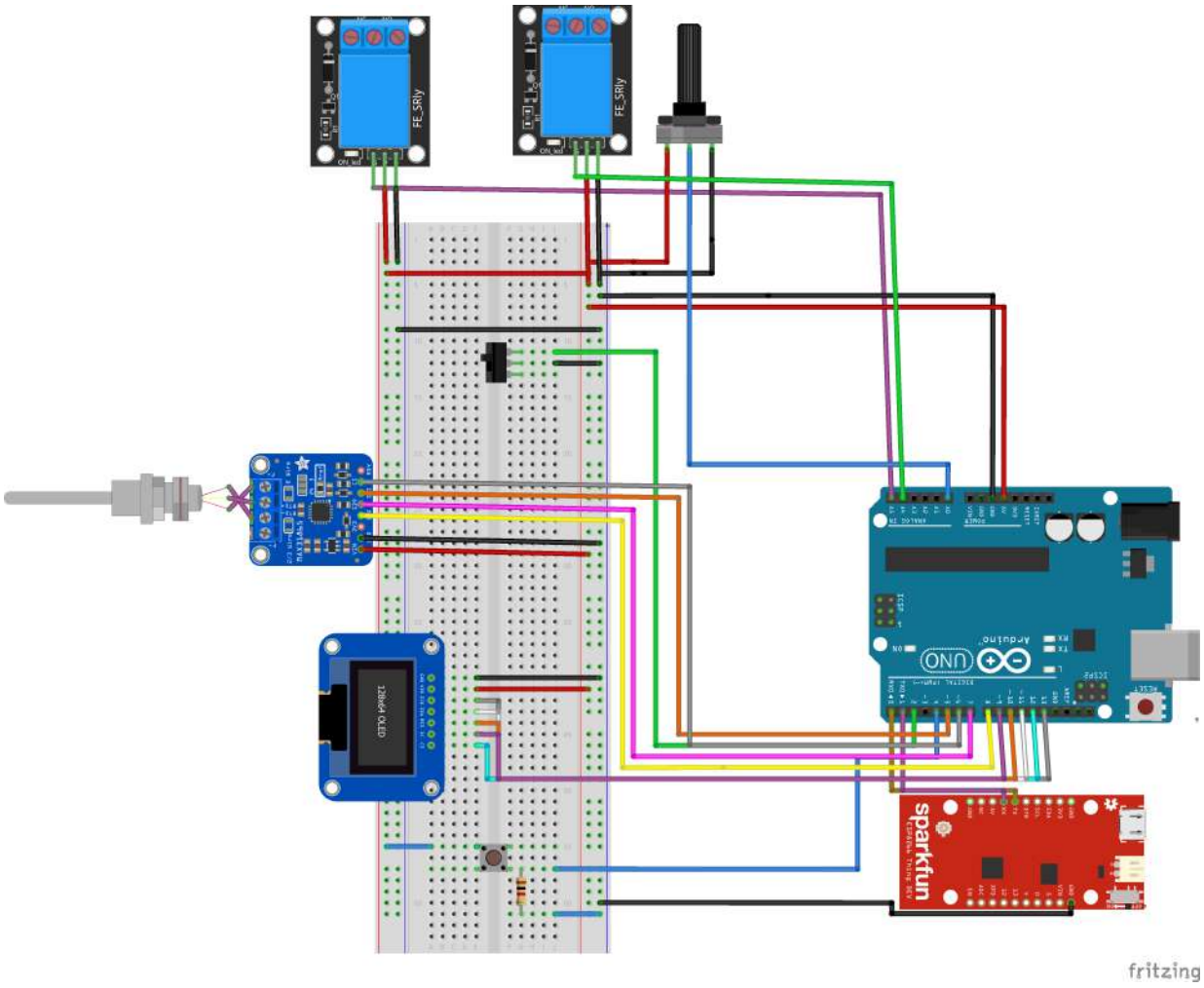


Immagine 10: Qui vediamo lo schema dei collegamenti tra i componenti utilizzati nel progetto.

6 Implementazione Fisica

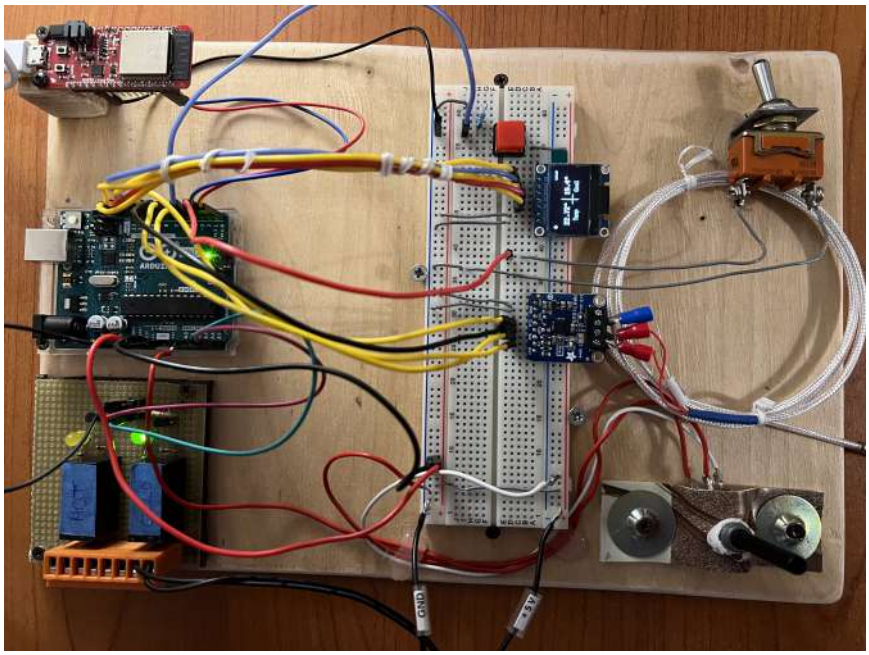


Immagine 11: Foto progetto finito.

Immagine 12: Schermata in Modalità "thermo on" che sta riscaldando (a sinistra) e Modalità "thermo off" (a destra).

Immagine 13: Schermata di scelta temperatura obiettivo in "thermo on"

7 Riferimenti

Per l'interfaccia dove viene impostata la temperatura goal ci siamo ispirati a questo progetto, riadattandone alcune parti in modo che fosse compatibile con i nostri range e i nostri componenti:
`https://www.youtube.com/watch?v=NPfaLKKsf_Q&list=PLSPeQu-8Pue2YBXg1Bcz_mh_08k9h_Fr3&index=2&t=2328s&ab_channel=upir`

Per la visualizzazione dei dati in Matlab abbiamo trovato tutte le informazioni utili sulla documentazione ufficiale:
`https://it.mathworks.com/help/matlab/index.html?s_tid=hc_panel`

Per l'utilizzo della libreria U8G ci siamo appoggiati alla documentazione fornita su github:
`https://github.com/olikraus/u8glib/wiki/userreference`

Per la ricerca di soluzioni ad eventuali problemi avuti con l'hardware ci siamo affidati ai vari forum su arduino:
`https://github.com/olikraus/u8glib/wiki/userreference`