Dovajanje odvečne solarne energije vodnemu grelcu s pomočjo Arduino dimmerja - poročilo

Marko Porenta

Kratek opis

Cilj projekta je izkoristiti čim večji delež odvečne solarne energije, ki jo lahko pridelamo z domačo sončno elektrarno. V sklopu tega projekta to dosežemo z dovajanjem odvečne energije vodnemu grelcu - na ta način presežek elektrike izkoristimo za gretje vode.

Pri proizvajanju energije s sončno elektrarno pogosto ne izkoristimo vsega potenciala, ki ga imajo solarni paneli - v primeru, da s sončno energijo pokrijemo vse energijske potrebe in prav tako izvažamo maksimalno količino energije v omrežje, preostale energije ne moremo izkoristiti - v stavbi v tem primeru ni več nobene naprave, ki ni ji lahko napeljali dodatno energijo.

Odvečno energijo želimo pridobiti iz sončnih celic s pomočjo segrevanja vode v vodnem grelcu - točno količino energije, ki mu jo bomo dovajali, bomo uravnavali s pomočjo dimmerja.

Prav tako je potrebno implementirati algoritem, ki uravnava količino energije, ki jo dovajamo vodnemu grelcu, saj nam količina dodatne energije ni znana, dokler dodatno ne obremenimo sončnih celic. Naloga algoritma je, da poskrbi za prižiganje grelca le v primeru, da sončne celice že v celoti energetske potrebe stavbe in jih želimo še dodatno izkoristiti.

Uporabljene komponente

- NodeMCU12E - ESP8266MOD WiFi Arduino modul
- Robotdyn AC light dimmer
- Napajalnik 5V
- NTC termistor, 10K Ω
    - NTC - (**N**egative **T**emperature **C**oefficient), upor pada ob višanju temperature

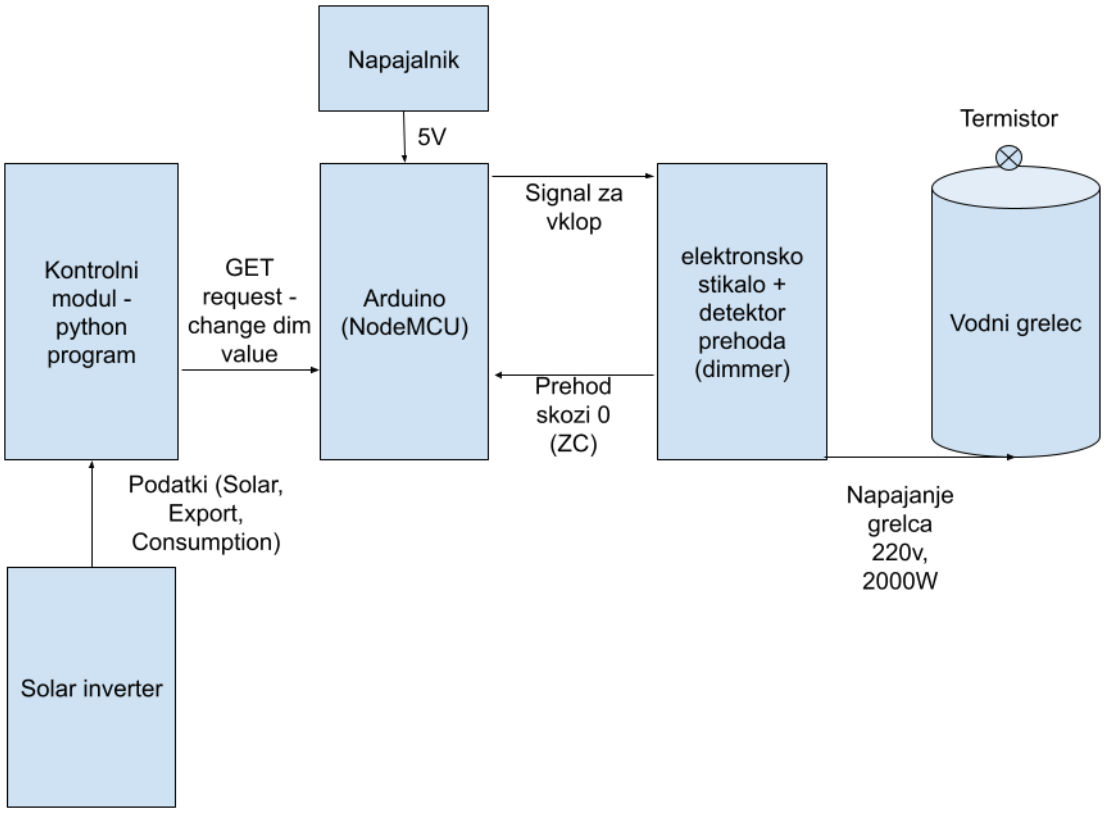
Projekt tudi predpostavlja domačo sončno elektrarno in vodni grelec s sledečimi parametri:

- Maksimalen izvoz sončne energije (**Export**): 3000W
- Energijska kapaciteta vodnega grelca (**Capacity**) : 2000W

Pri drugačnih konfiguracijah sončne elektrarne/vodnega grelca je te vrednosti mogoče ponastaviti v Python programu.

Diagram delovanja

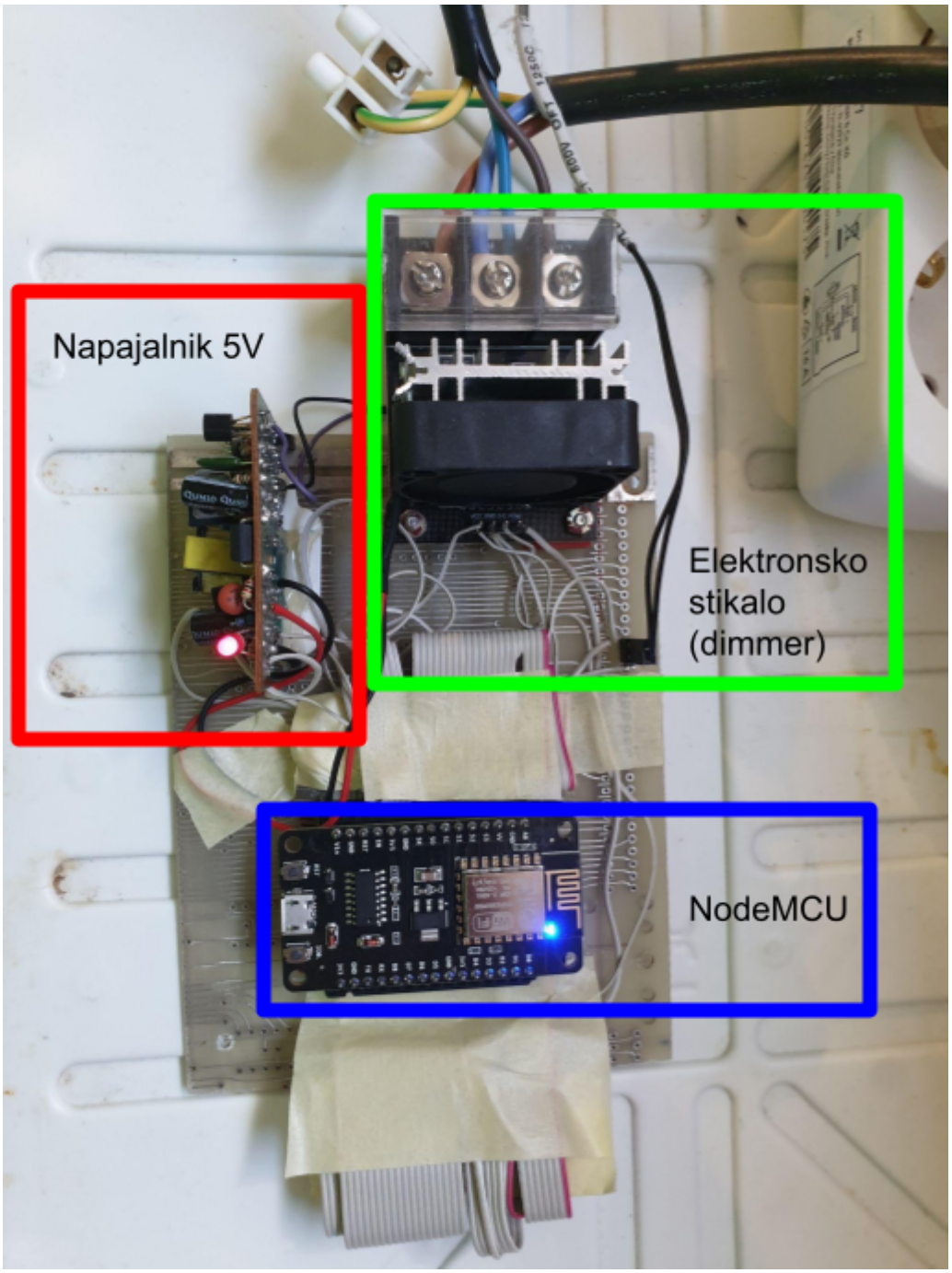
Na spodnji sliki je preprosta shema sestavnih delov projekta.



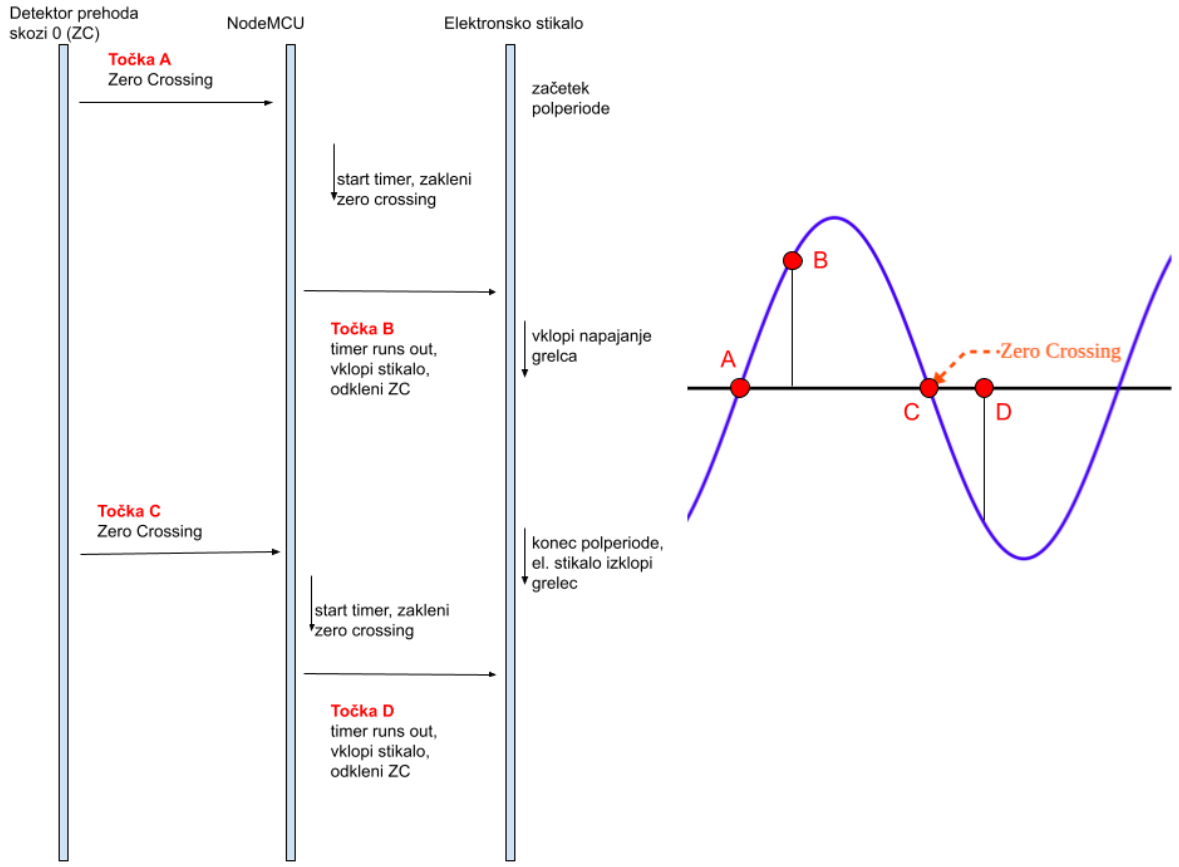
Vsaka komponenta sistema ima določene naloge:

- Solarni inverter
    - Periodično pošilja osnovne podatke o delovanju solarnega omrežja Python programu (Solar, Export, Consumption) preko Modbus protokola
- Kontrolni modul - Python program
    - Na podlagi prejetih podatkov s preprostim algoritmom izračuna vrednost, na katero želimo nastaviti dimmer - to preko GET zahtevka sporoči Arduinu
- Arduino(NodeMCU)
    - Poganja preprost spletni strežnik, na katerem lahko vidimo trenutno temperaturo grelca in vrednost dimmerja; preko strežnika lahko tudi ročno nastavimo vrednost dimmerja
    - Preko GET zahtevkov prejema vrednost, na katero želimo nastaviti dimmer, in mu pošlje želeni signal
- elektronsko stikalo + detektor prehoda (dimmer)
    - Prejema vrednosti v razponu 0-255, nato glede na prejeto vrednost določi zakasnitev, po kateri vključi/izključi dovod elektrike vodnemu grelcu (glej poglavje Zero Crossing)
- Napajalnik
- Termistor

Slika vezja



Zero Crossing



Mehanizem, s katerim dimmer določi količino električne napetosti, ki jo prejme vodni grelec, se imenuje Zero Crossing - Ob vsakem prehodu sinusa čez ničelno točko se v Arduino kodi izvede funkcija `zcDetectISR`:

```
void zcDetectISR() {

  if (zcPending == 0) {

      if (tarBrightness < 256 && tarBrightness > 0) {
      zcPending = 1;
      int dimDelay = 10000 * lindelay[tarBrightness];
      hw_timer_arm(dimDelay);
      }

  }
}
```

Funkcija na podlagi prejete vrednosti dimmerja določi časovni zamik, preden vklopi signal, ki doseže vodni grelec - s tem regulira količino energije, ki jo ta prejme. Funkcijo prav tako zaklene, dokler se ta v celoti ne izvede - s tem preprečimo napačno večkratno nastavljanje timerja v primeru potencialnih motenj signala.

Arduino program

S kodo na NodeMCU12E modulu želimo doseči:

- Vzpostavitev spletnega strežnika, do katerega lahko dostopamo preko določenega naslova in spremljamo trenutno stopnjo dimmerja in temperaturo vodnega grelca; po zgledu osnove za ESP8266 spletni strežnik s statičnim naslovom
- branje vrednosti, ki mu jih preko GET requesta pošilja Python program, in nastavitev dimmerja na to vrednost;
    - Python program izračuna vrednost, na katero moramo nastaviti dimmer, in jo preko GET requesta posreduje Arduino programu; ta nastavi dimmer na novo vrednost in tako uravnava količino energije, ki jo dovajamo vodnemu grelcu
- branje temperature s termistorja
    - s termistorjem merimo temperaturo vodnega grelca in jo izpisujemo na strani web serverja

---

Poleg vgrajenih so za delovanje ESP8266 modula in dimmerja potrebne še knjižnice:

- ESP8266WiFi (del jedrnih ESP8266 knjižnic)
- ESPASyncTCP
- ESPASyncWebServer
- ESP8266-wifi-light-dimmer

ESP8266 Web server

Najprej implementiramo web server, ki bo tekel na našem NodeMCU modulu:

```
// Import required libraries
#include <ESP8266WiFi.h>
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <hw_timer.h>

// network credentials
const char* ssid = "imeSSID"; // ACCESS POINT
const char* password = "gesloSSID"; //ENTER PASSWORD IF DESIRED

// Set your Static IP address
IPAddress local_IP(192, 168, 58, 95);
// Set your Gateway IP address
IPAddress gateway(192, 168, 58, 1);

IPAddress subnet(255, 255, 255, 0);
IPAddress primaryDNS(192, 168, 58, 1);   //optional
IPAddress secondaryDNS(192, 168, 58, 1); //optional

// Create AsyncWebServer object on port 80
AsyncWebServer server(80);
```

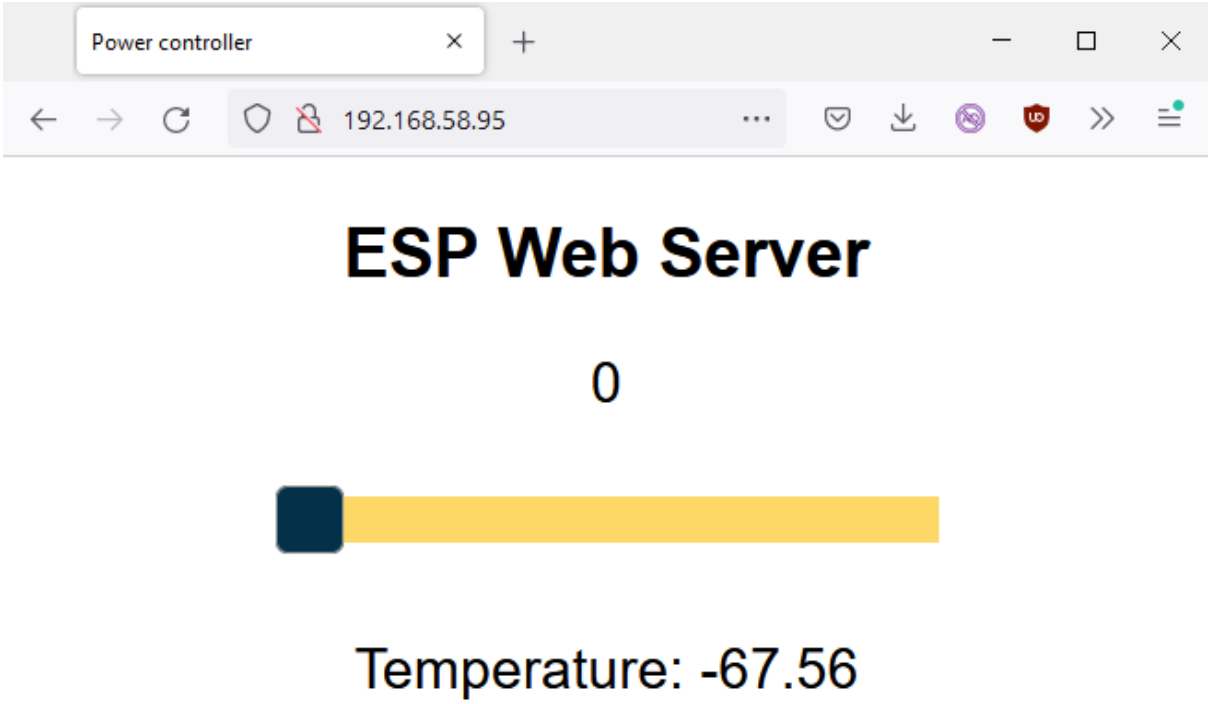
ter znotraj začetne `setup` funkcije:

```
void setup(){
// Serial port for debugging purposes
Serial.begin(115200);

//WiFi.softAP(ssid, password);
//IPAddress IP = WiFi.softAPIP();
//Serial.println(IP);

// Configures static IP address
  if (!WiFi.config(local_IP, gateway, subnet, primaryDNS, secondaryDNS))
{
      Serial.println("STA Failed to configure");
  }

  // Connect to Wi-Fi network with SSID and password
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
```

```
      delay(500);
      Serial.print(".");
  }
  // Print local IP address and start web server
  Serial.println("");
  Serial.println("WiFi connected.");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  //server.begin();

// Route for root / web page
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
      readThermistor();
      request->send_P(200, "text/html", index_html, processor);
});

// Send a GET request to <ESP_IP>/slider?value=
server.on("/slider", HTTP_GET, [] (AsyncWebServerRequest *request) {
  String inputMessage;

  readThermistor();

  // GET input1 value on <ESP_IP>/slider?value=
  if (request->hasParam(PARAM_INPUT)) {
      inputMessage = request->getParam(PARAM_INPUT)->value();
      sliderValue = inputMessage;
  } else {
      inputMessage = "No message sent";
  }

  Serial.print("Slider: "); Serial.println(inputMessage);
  request->send(200, "text/plain", "OK");

  Serial.print("Temperature:"); Serial.println(temperature);
});

// Start server
server.begin();

}
```

Ob zagonu NodeMCU deluje kot spletni strežnik s statičnim naslovom, ki bo preko našega domačega omrežja sprejemal GET zahteve, ki jih bo prejemal s strani Python programa. Ob vsaki prejeti GET zahtevi strežnik osveži stran in nam prikaže trenutno stopnjo dimmerja ter temperaturo vodnega grelca, ki jo merimo s termistorjem.

Ob zagonu kode na Arduinu dobimo preprosto spletno stran:



Na voljo imamo drsnik, ki nam omogoči ročno spreminjanje vrednosti dimmerja med 0 in 255. Trenutno nam stran še ne kaže pravilne temperature - to bomo lahko izmerili, ko Arduino modul povežemo z vodnim grelcem in termistorjem.

Branje temperature s termistorjem

Da spremljamo delovanje našega programa in temperaturo našega grelca, bomo našemu vezju dodali termistor - podrobnosti so na voljo v sledečem zgledu.

```
void readThermistor()
{
  //Thermistor readings
  double Vout, Rth, adc_value;
  adc_value = analogRead(A0);
  Vout = (adc_value * VCC) / adc_resolution;
  Rth = (VCC * R2 / Vout) - R2;

/*  Steinhart-Hart Thermistor Equation:
 *  Temperature in Kelvin = 1 / (A + B[ln(R)] + C[ln(R)]^3)
 *  where A = 0.001129148, B = 0.000234125 and C = 8.76741*10^-8  */
  temperature = (1 / (A + (B * log(Rth)) + (C * pow((log(Rth)),3))));
// Temperature in kelvin
  temperature = temperature - 273.15;  // Temperature in degrees celsius
  Serial.print("Temperature calc:"); Serial.println(temperature);

}
```

Z funkcijo `readThermistor` v Arduino programu temperaturo ponovno preberemo ob vsakem prejetem GET zahtevku - ta temperatura se shrani v spremenljivko **temperature**, ki jo nato prikažemo na strani Arduino strežnika.

Izgled vezja, ročno nastavljanje dimmerja, izpis temperature - video

Python program

Naš Python program opravlja več funkcij:

- Pridobi podatke o pridelani sončni energiji (spremenljivka **Solar**), trenutni porabi energije (spremenljivka **Consumption**) in izvozu sončne energije v omrežje (spremenljivka **Export**) - naš cilj je, da najprej zagotovimo, da izvažamo vso električno energijo, ki jo lahko; le v tem primeru imamo odvečno energijo, ki jo lahko z grelcem izkoristimo
- Program na podlagi teh vrednosti izračuna, koliko energije smemo dovajati vodnemu grelcu; željeno vrednost dimmerja nato preko GET requesta posreduje Arduinu

---

Poleg vgrajenih Python knjižnic za branje podatkov z solarnega inverterja potrebujemo zunanjo knjižnico solaredge_modbus.

Program je v grobem razdeljen na objekte **Controller**, **Dimmer** in **Inverter**, ter glavni del **boilerOOP.py** - ta izvira iz predloge example.py, ki služi kot primer preprostega programa, ki se poveže na inverter in z njega bere želene informacije.

Začnemo z dodajanjem zahtevanih knjižnic, vnosom parametrov in povezavo na Inverter (**IP naslov in port Inverterja programu navedemo kot vhodne argumente poleg stikala --div, npr.** `192.168.58.33 1302 -div`):

```
#!/usr/bin/env python3
import sys
import argparse
import json
import time
from datetime import datetime

import solaredge_modbus

from Controller import Controller
from Dimmer import Dimmer
from Inverter import Inverter

controller = Controller("192.168.58.95", 60) #ip, max temperature
dimmer = Dimmer(2000) #dimmer capacity in watts
inverterM = Inverter(3000, 400) # export limit in watts, reserve

argparser = argparse.ArgumentParser()
argparser.add_argument("host", type=str, help="Modbus TCP address")
argparser.add_argument("port", type=int, help="Modbus TCP port")
argparser.add_argument("--timeout", type=int, default=1, help="Connection
timeout")
argparser.add_argument("--unit", type=int, default=1, help="Modbus device
address")
argparser.add_argument("--json", action="store_true", default=False,
help="Output as JSON")
argparser.add_argument("--div", action="store_true", default=False, help="Output
for diverter")
args = argparser.parse_args()

inverter = solaredge_modbus.Inverter(
    host=args.host,
    port=args.port,
    timeout=args.timeout,
    unit=args.unit
)
maxexport = -999999
maxexporttime = datetime.now()
export_limit = 2700
```

tu tudi nastavimo željene parametre objektov:

- Controller
    - IP naslov do Arduino strežnika, kateremu bomo pošiljali podatke (192.168.58.95)
    - maksimalna dovoljena temperatura vodnega grelca (60°C)
- Dimmer

    - kapaciteta našega vodnega grelca v vatih (2000W)
- Inverter
    - maksimalna možna količina izvažane energije v vatih (3000W)
    - rezerva - program dopušča izvažanje energije, ki je za največ rezervno količino manjše od maksimuma (npr. maksimum = 3000W, rezerva = 400W → program dopušča tudi izvažanje energije med 2600W in 2999W)

Preko inverterja nato beremo podatke o pridelani sončni energiji - za potrebe naloge so bistveni:

- pridelana sončna energija (spremenljivka **Solar**)
- trenutna poraba energije (spremenljivka **Consumption**)
- izvoz sončne energije v omrežje (spremenljivka **Export**)

Na podlagi teh podatkov bomo opazovali, koliko energije nam ostane za gretje vode.

```
while(True):
   values = {}
   values = inverter.read_all()
   meters = inverter.meters()
   batteries = inverter.batteries()
   values["meters"] = {}
   values["batteries"] = {}

   for meter, params in meters.items():
       meter_values = params.read_all()
       values["meters"][meter] = meter_values

   for battery, params in batteries.items():
       battery_values = params.read_all()
       values["batteries"][battery] = battery_values

   if args.json:
       print(json.dumps(values, indent=4))
       break
   elif args.div:
      now = datetime.now()
      dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
      print("-------------- ", dt_string)

      solar = (values['power_ac'] * (10 ** values['power_ac_scale'])) #power_ac,
power_ac_scale -> values
      #print(f"\tSolar power: {solar}W")
      for k, v in meter_values.items():
        if k == 'power':
          calculateDimValue(solar, -v, solar + v)
          break
   sys.stdout.flush()
   sys.stderr.flush()

   time.sleep(3)
```

V 3-sekundnih intervalih s pomočjo knjižnice beremo podatke in jih podajamo funkciji `calculateDimValue`:

```
def calculateDimValue(solar, consumption, export):

   print("Solar: %i W" % (solar))
   print("Total Consumption: %i W" % (consumption))
   print("Export: %i" % (export))

   #controller.temperature = controller.updateTemperature(controller.ip)

   #export is at max/over max:
   if(export >= inverterM.exportmax):
       a = 255 - dimmer.value
       dimmer.updateValue(dimmer.value + int(a/2))
       controller.updateDimmer(controller.ip, dimmer.value)
   else: #export is under max
       dimmer.updateValue(max(0, dimmer.value + (export - (inverterM.exportmax -
inverterM.reserve)) / int(dimmer.maxWattage / 255)))
       controller.updateDimmer(controller.ip, dimmer.value)
   print("Changing dimmer value to %i:" % (dimmer.value))
   print("-----------")
```

Ta funkcija s preprostim algoritmom preveri trenutno stanje izvoza energije:

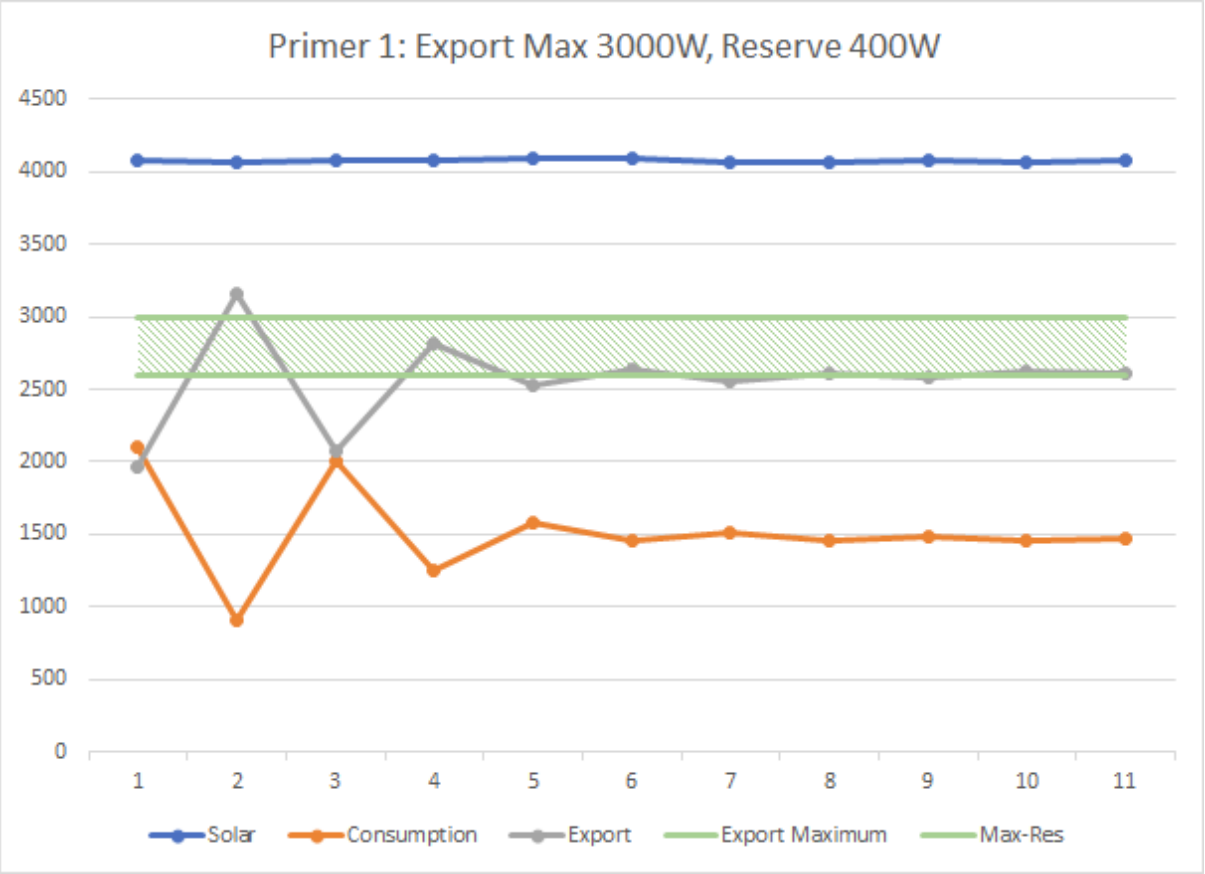
- Če proizvajamo dovolj elektrike, da lahko dosežemo maksimalno mejo izvoza energije, imamo odvečno energijo, ki jo lahko dovajamo vodnemu grelcu;
    - stopnjo vodnega grelca vedno višamo za polovico maksimalne možne spremembe (npr. 0 → 127, 127 → 190,...) ; s tem omejimo možnost, da grelcu dodelimo preveč energije in s tem zmanjšamo izvoz energije v omrežje
- V primeru, da je izvoz energije pod maksimumom, lahko izračunamo točno stopnjo, na katero je potrebno nastaviti dimmer, da bomo v naslednjem koraku izvažali maksimum

Dejansko pošiljanje stopnje dimmerja, na katero želimo nastaviti grelec vode, poteka v funkciji `controller.updateDimmer`:

```
def updateDimmer(self, ip, newValue):
   try:
       updateDimmer = requests.get('http://%s/slider?value=%i' % (ip, newValue),
timeout=10)
       #print(updateDimmer.text)
   except requests.exceptions.Timeout as e:
       print(e)
```

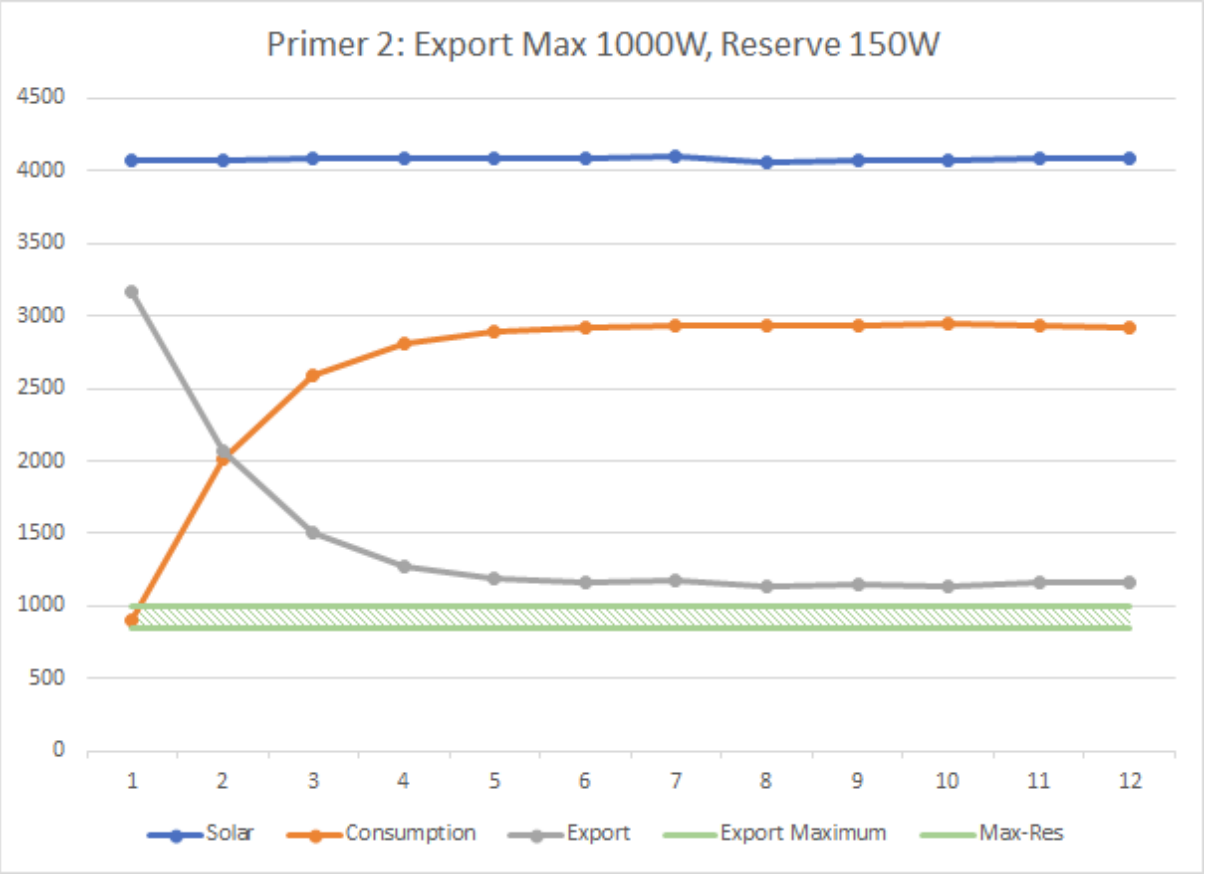
Funkcija kot parametra prejme IP naslov Arduina in novo vrednost, na katero želimo nastaviti dimmer.

Primer 1 - Export Max 3000W, Reserve 400W



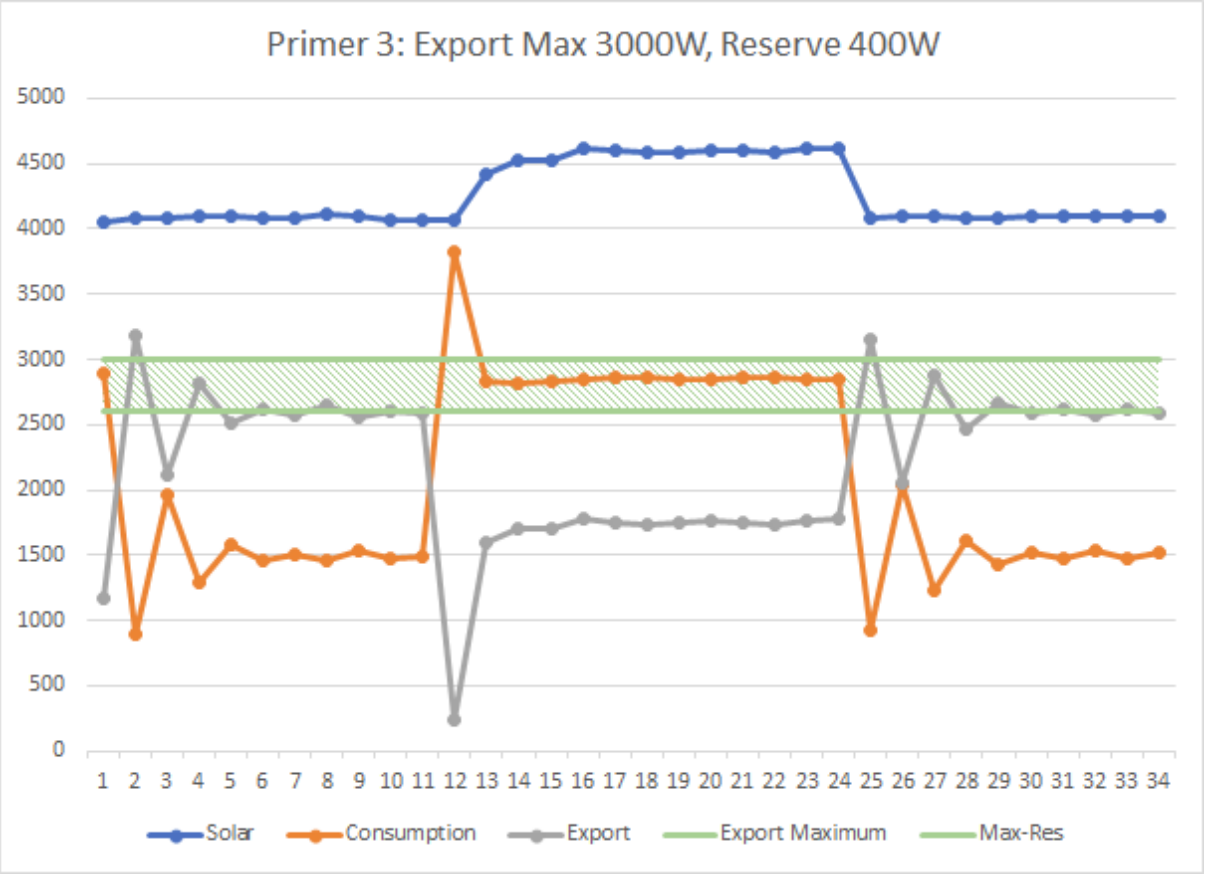
V tem primeru sta pridelava sončne energije in energijska poraba hiše dokaj enakomerna, zato algoritem nima nobenih težav z doseganjem željene meje izvažanja. Spremembe v porabi lahko pripišemo spreminjanju količine energije, ki jo prejema vodni grelec.

Primer 2 - Export Max 1000W, Reserve 150W



V drugem primeru nam kljub maksimalnemu povečanju energije, ki jo dovajamo grelcu, ostane več energije, kot smo je zmožni izvažati v omrežje.

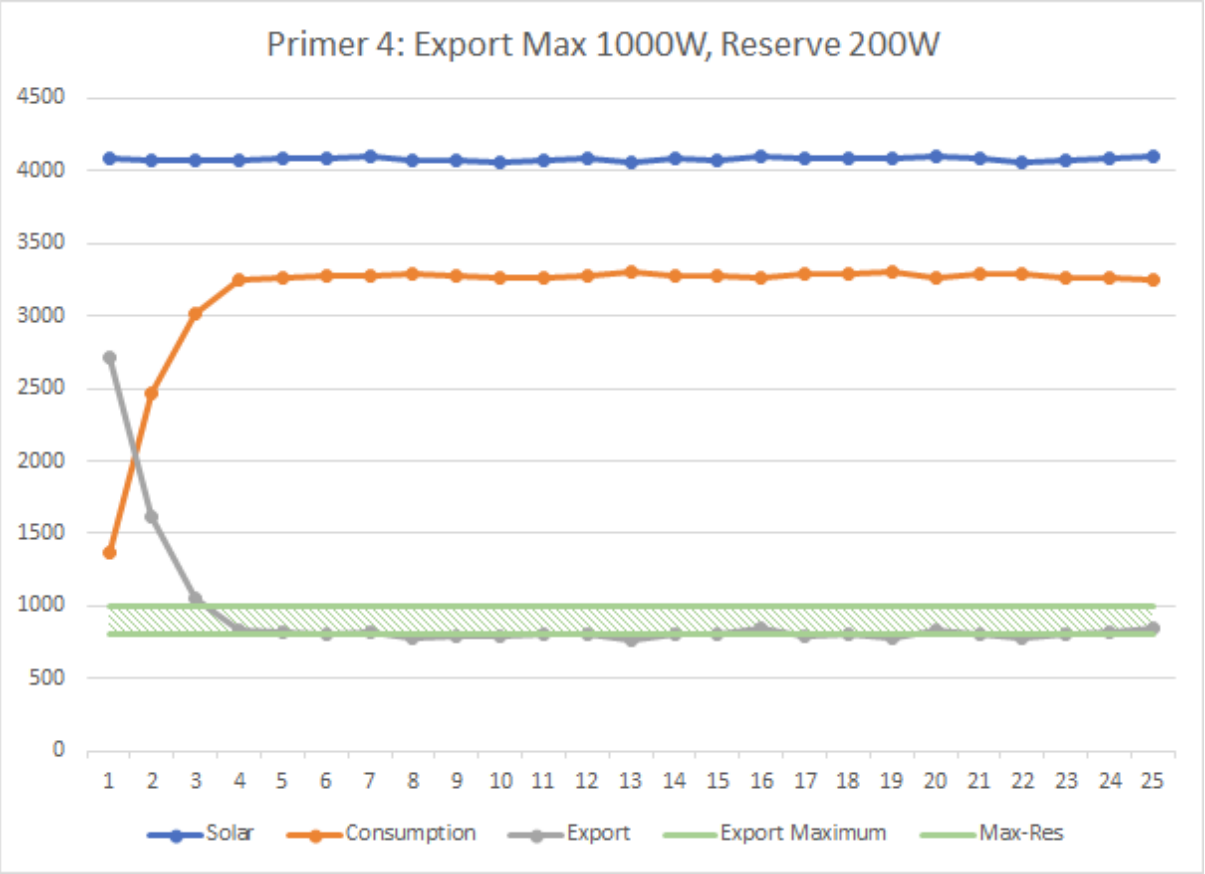
Primer 3 - Export Max 3000W, Reserve 400W



V tretjem primeru se algoritem prvič sreča z nepričakovano povečavo v porabi energije (med korakom 11 in korakom 12). Tu se poleg porabe poveča tudi količina energije, ki smo jo dobili iz sončnih celic, kar je smiselno - ne vemo točno, koliko energije so celice zmožne pridelati v danem trenuktu, dokler je ne poskusimo porabiti. Ko v tem koraku pride do povečane porabe, lahko izračunamo točno, na katero stopnjo dimmerja moramo ponastaviti vodni grelec - v naslednjem koraku se spremenljivka Export ponovno vrne na želeno vrednost.

Ko se med korakoma 24 in 25 naprava, ki je povzročila dvig porabe, izklopi, algoritem ponovno lahko počasi viša vrednost dimmerja in s tem koristno porabi trenutno odvečno energijo.

Primer 4 - Export Max 1000W, Reserve 200W



[Primer 4 - video]