Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Un sistema di domotica per il controllo energetico domestico con tecnologie open-source.

Relatore:
Chiar.mo Prof.
Renzo Davoli

Presentata da:
Francesco Bartolini

Sessione II
Anno Accademico 2015/2016

Sommario

Data la sempre maggiore richiesta di fabbisogno energetico, si è sviluppata una nuova filosofia nella gestione dei consumi energetici, il *DSM (demand side management)*, che ha lo scopo di incoraggiare il consumatore ad usare l'energia in modo più intelligente e coscienzioso. Questo obiettivo, unito all'accumulo di energia da fonti rinnovabili, permetterà un abbassamento dell'utilizzo dell'energia elettrica proveniente dal consumo di fonti non rinnovabili e altamente inquinanti come quelle a combustibili fossili ed una diminuzione sia del consumo energetico, sia del costo per produrre energia che dell'energia stessa. L'*home automation* e la domotica in ambiente domestico rappresentano un esempio di DSM.
L'obiettivo di questa tesi è quello di creare un sistema di home automation utilizzando tecnologie open-source. Sono stati utilizzati device come board Arduino UNO, Raspberry Pi ed un PC con sistema operativo GNU/Linux per creare una simulazione di un sistema di home automation abbinato alla gestione di celle fotovoltaiche ed energy storaging.
Il sistema permette di poter spegnere un carico energetico in base a delle particolari circostanze come, per esempio, il superamento di una certa soglia di consumo di energia elettrica. Il software utilizzato è open-source e mira a poter ottimizzare il consumo energetico secondo le proprie finalità. Il tutto a dimostrare che si può creare un sistema di home automation da abbinare con il presente e futuro delle fonti rinnovabili utilizzando tecnologie libere in modo tale da preservare privacy e security oltre che customizzazione e possibilità di adattamento a diverse circostanze.
Nella progettazione del sistema è stato implementato un algoritmo per gestire varie situazioni all'interno di un ambiente domestico. La realizzazione di tale algoritmo ha prodotto ottimi risultati nella raggiungimento degli obiettivi prefissati. Il progetto finale di questa tesi può essere ulteriormente ampliato ed il codice, sotto licenza GPLv3, è reperibile in un repository pubblico.

Indice

1 Lo Stato dell'Arte **1**
1.1 Introduzione 1
1.2 Il software come soluzione 5

2 Implementazione **9**
2.1 Strumentazione 10
2.1.1 Hardware 11
2.1.2 Software 12
2.2 Schema Generale 13
2.2.1 Il simulatore 14
2.2.2 RaspberryPi 18
Il Database 19
Studio e progettazione dell'algoritmo 21
2.2.3 Arduino 26
2.2.4 Configurazioni di rete e configurazioni varie 28

3 Scenari e Casi d'Uso **31**
3.1 Scenario 1 31
3.2 Scenario 2 33
3.3 Scenario 3 33
3.4 Scenario 4 36

4 Evoluzioni future e conclusioni **39**

Allegati **42**

Elenco delle figure

1.1 Produzione elettrica teorica del fotovoltaico. 2
1.2 Top 10 per capacità annuale e cumulativa totale. 2
1.3 Dati sul consumo energetico generale in Italia tra settori. Confartigianato. 3
1.4 Dati sul consumo energetico generale in Italia tra settori, confronto tra gli anni 2003 e 2013. Confartigianato. 3
1.5 Schema logico funzionamento MQTT. 6

2.1 Schema logico-implementativo: in verde le parti simulate, in arancione quelle non simulate. 13
2.2 Struttura Simulatore. 14
2.3 Schema logico del funzionamento dei moduli del simulatore. . 15
2.4 Schema della comunicazione tra componenti. 17
2.5 Struttura Rpi. 18
2.6 Interazione tra moduli e database 20
2.7 Configurazione Network: in verde connessione wireless, in blu connessione wired. W = interfaccia Wireless, E = interfaccia Ethernet. 28

3.1 Scenario di scarico sulla batteria e notifica nel file di log. . . . 31
3.2 Scenario di carico sulla batteria. 32
3.3 Scenario di carico sulle celle fotovoltaiche e caricamento della batteria. 33
3.4 Scenario con consumo al limite dei carichi *non* interrompibili. 34
3.5 Scenario con interruzione del carico interrompibile. 34
3.6 Scenario con interruzione del carico interrompibile, mostrando anche il grafico del consumo totale. 35

3.7 Scenario con livello dei consumi nei limiti e successiva accensione del carico interrompibile. 36
3.8 Altro esempio di scenario con livello dei consumi nei limiti e successiva accensione del carico interrompibile. 36

4.1 In blue la potenza reale (100 Watt) ed in rosso la stima dell'algoritmo 1 46
4.2 In blue la potenza reale (100 Watt) ed in rosso la stima dell'algoritmo 2 47
4.3 In blue la potenza reale (100 Watt) ed in rosso la stima dell'algoritmo 3 49
4.4 In blue la potenza reale (100 Watt) ed in rosso la stima dell'algoritmo 4 50
4.5 Schema di collegamento del contatore ad impulsi. 54
4.6 Collegamento del cavo elettrico con il contatore. In blu il cavo neutro, in marrone la fase. 55
4.7 Resistenza da 10 $k\Omega$. 55
4.8 Collegamento dal foro 21 al GND di Arduino. 56
4.9 Tagliare e saldare come in schema. 56
4.10 Schema finale. 57
4.11 Schema collegamento. 58

Elenco delle tabelle

2.1 Lista elementi hardware. 11

Capitolo 1

Lo Stato dell'Arte

1.1 Introduzione

Una delle problematiche più in evidenza dell'era moderna è l'incremento costante di fabbisogno energetico. In un mondo sempre più tecnologico, con una maggiore presenza di apparati elettronici nella vita di tutti i giorni, la gestione sapiente dell'energia elettrica risulta essere uno dei fattori più importanti sia in ottica di conservazione ambientale che dal punto di vista più materiale del risparmio economico.
Il *demand side management (DSM)* [1] identifica una nuova filosofia di gestione dei consumi energetici, che ha l'obiettivo di incoraggiare il consumatore ad usare meno energia soprattutto durante gli orari di picco. In tal modo si può ridurre al minimo necessario il valore della potenza impegnata, cioè richiedere un valore minimo di disponibilità alla produzione di energia elettrica. Ciò vuol dire non solo consumare meno energia in generale, ma anche evitare costi aggiuntivi nel pianificare, programmare e costruire impianti e reti tecnologiche per la gestione dell'aumento dei carichi energetichi soprattutto in questi orari. L'idea presente nel *DSM* è quindi quella di posticipare l'uso di energia elettrica dagli orari di picco ad un altro orario, magari notturno [2]. Tutto ciò può essere all'avanguardia se si introduce l'accumulo di energia da fonti rinnovabili ed il conseguente abbassamento dell'utilizzo dell'energia elettrica proveniente dal consumo di fonti non rinnovabili e altamente inquinanti come quelle a combustibili fossili. Si osservano oggigiorno sempre più esempi di utilizzo di fonti rinnovabili non solo nella produzione dell'energia elettrica per le abitazioni domestiche, ma anche in altri settori come quello industriale, del riscaldamento e dei trasporti, come esposto dall'annuale report di Ren21 (Renewable energy policy network for the 21st century) [3]. In generale il 2015 è stato l'anno in cui gli investimenti hanno toccato il livello più alto

di sempre, secondo quanto riporta l'annuale rapporto di *Bloomberg Energy Finance* [4]. Dal report *Photovoltaic Power System Programme* dell'agenzia internazionale dell'Energia (IEA PVPS), l'Italia nel 2014 è risultata essere il paese con il più alto contributo al mondo di fotovoltaico nella domanda elettrica [5].

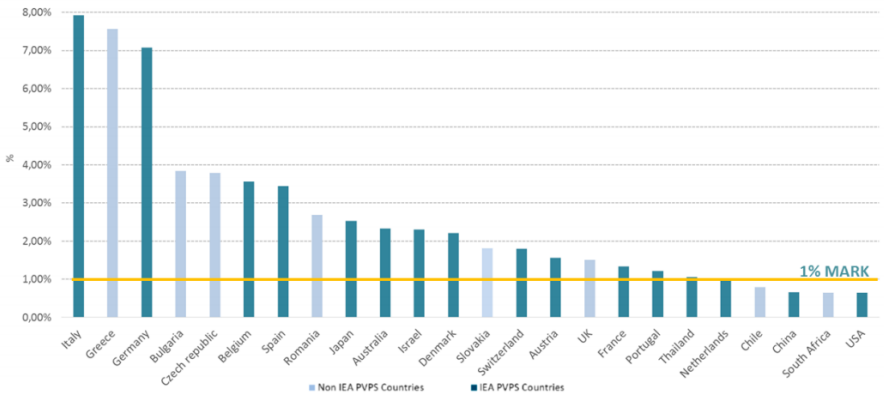

Figura 1.1: Produzione elettrica teorica del fotovoltaico.

Sempre secondo lo stesso snapshot, in Italia il contributo del fotovoltaico alla domanda di energia elettrica è pari al 7,9%, con il contributo fotovoltaico europeo complessivo pari al 3,5% della domanda di energia elettrica dell'Europa [5]. Inoltre il nostro Paese, sempre secondo il report, nel 2014 è quarto nella classifica mondiale per capacità cumulativa installata (ovvero la somma parziale, per periodi consecutivi, di tutte la capacità o quantità di una data risorsa, in questo caso energia elettrica).

	TOP 10 COUNTRIES IN 2014 FOR ANNUAL INSTALLED CAPACITY		TOP 10 COUNTRIES IN 2014 FOR CUMULATIVE INSTALLED CAPACITY	
1st	China	10,6 GW	Germany	38,2 GW
2nd	Japan	9,7 GW	China	28,1 GW
3rd	USA	6,2 GW	Japan	23,3 GW
4th	UK	2,3 GW	Italy	18,5 GW
5th	Germany	1,9 GW	USA	18,3 GW
6th	France	0,9 GW	France	5,7 GW
7th	Australia	0,9 GW	Spain	5,4 GW
8th	Korea	0,9 GW	UK	5,1 GW
9th	South Africa	0,8 GW	Australia	4,1 GW
10th	India	0,6 GW	Belgium	3,1 GW

Figura 1.2: Top 10 per capacità annuale e cumulativa totale.

In Italia, secondo GSE (Gestore dei Servizi Energetici, ex società Gestore della Rete di Trasmissione Nazionale S.p.a.), 630.192 impianti fotovoltaici su 648.418, ovvero il 97,2%, sono collegati alla rete di bassa tensione (quindi collegati a reti domestiche) [6].

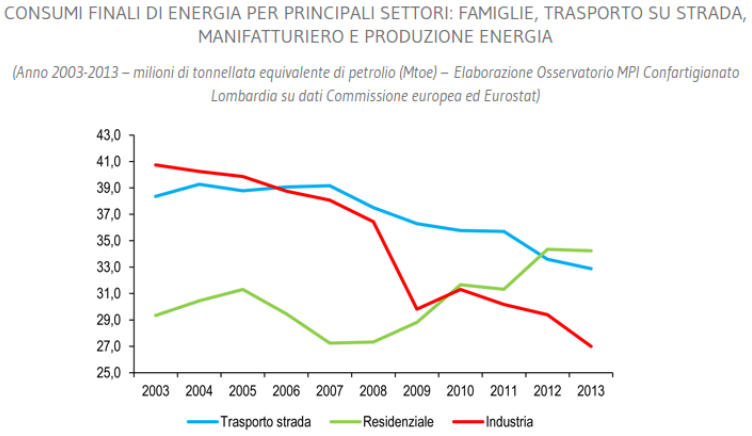

Figura 1.3: Dati sul consumo energetico generale in Italia tra settori. Confartigianato.

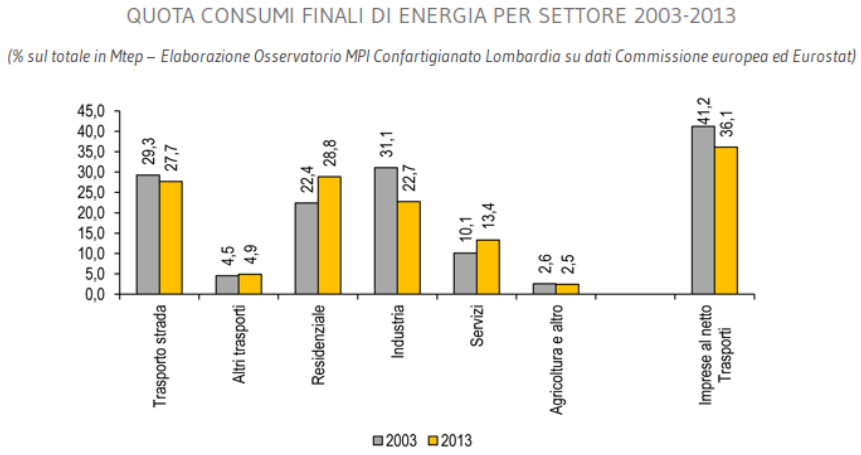

Figura 1.4: Dati sul consumo energetico generale in Italia tra settori, confronto tra gli anni 2003 e 2013. Confartigianato.

Dalle figure 1.3 and 1.4 possiamo notare come il nostro Paese risulta essere più *energivoro* per quanto riguarda il consumo residenziale (28.8 %) piuttosto che quello industriale (22,7 %) o dei trasporti di strada(27,7 %),

dall'Industria (22,7%), dai Servizi (13,4%), dagli altri trasporti (ferrovia, aereo e nave con il 4,9%) e Agricoltura e altro (2,5%), come spiegato da un rapporto di Confartigianato con dati provenienti dalla Commissione Europea ed Eurostat) del 2013[7].
Da rimarcare la diminuzione del rapporto tra l'energia consumata e il valore aggiunto prodotto delle imprese, avvenuta tra l'anno 2003 e 2013, passata dal 41,2% al 36.1%.

Ciò significa che l'industria tende ad attrarre con attenzione tutte le politiche di contenimento dei consumi. Perciò appare sempre più importante l'intervento della razionalizzazione di consumi in ambito residenziale, soprattutto per invertire la tendenza alla crescita in atto negli ultimi 5 anni. Ciò va fatto con una attenzione particolare all'utilizzo delle fonti rinnovabili, quando disponibili direttamente su posto. Proprio in tal caso l'applicazione delle metodiche *DSM*, danno il massimo risultato.

1.2 Il software come soluzione

Il recente elevato e veloce sviluppo tecnologico attorno alla domotica ed all'automazione in ambiente domestico, ha portato a quello che viene chiamato *Internet of Things* (IoT). Secondo tale approccio si auspica che ogni device presente all'interno e non di un'abitazione, edificio, mezzo di trasporto venga interconnesso in reti più o meno grandi, in modo tale da far sì che tutti questi oggetti, collegati tra loro e raggiungibili da qualunque parte del globo, possano comunicare e scambiarsi dati [8].

Nel mercato sono presenti molte aziende che sviluppano soluzioni non open-source per la gestione di apparati domotici, basti pensare a Nest Labs [9] e Honeywell [10]. In rete è possibile trovare anche tantissime soluzioni che possono permettere a vari tipi di device diversi di *parlare* tra di loro grazie a protocolli che permettono di stabilire degli standard di comunicazione in ambiente domestico, come per esempio KNX [11] e ZigBee [12], conferendo delle basi solide per lo sviluppo di impianti domestici con software open-source.

Per quanto riguarda gli ambienti domestici, un software open-source molto famoso è *OpenHAB*. Si tratta di un software per l'integrazione di diversi sistemi e tecnologie di automazione domestica in un'unica soluzione, offrendo uniformità di interfacce utente e di regole di automazione [13].
Altri esempi di *home automation* possono riguardare *CalaOS*[14], software open-source con licenza GPLv3 predisposto per controllare e monitorare un'abitazione domestica al fine di trasformarla in una *smart home*.
Domoticz[15] è un software di automazione domestica con un largo supporto di device che variano da stazioni meteorologiche a rilevatori di fumo, fino a controlli da remoto. Il tutto con un'interfaccia `HTML5`, rendendo perciò il tutto accessibile sia da PC che dai più moderni smartphone.
Un'altra piattaforma di *home automation*, basata anch'essa su software open-source e licenza *MIT*, è *Home Assistant*[16]. Sfrutta `Python3` e *Docker*[17] al fine di essere portabile in diversi dispositivi da RaspberryPi ad un semplice NAS.
Più complessa invece è l'offerta proposta da *Open Motics*[18]. Utilizza software ma anche hardware open-source, proponendo moduli hardware da installare nella propria abitazione, concentrandosi così più su soluzioni *hardwired*.

Sono presenti anche *messaging protocol* come MQTT [19], progettato per dispositivi vincolati a bassa larghezza di banda, alta latenza o reti inaffidabili, ideale per la comunicazione *machine-to-machine* (M2M) e quindi per l'IoT e per device dove il consumo di batteria e di banda deve essere ridotta al minimo.

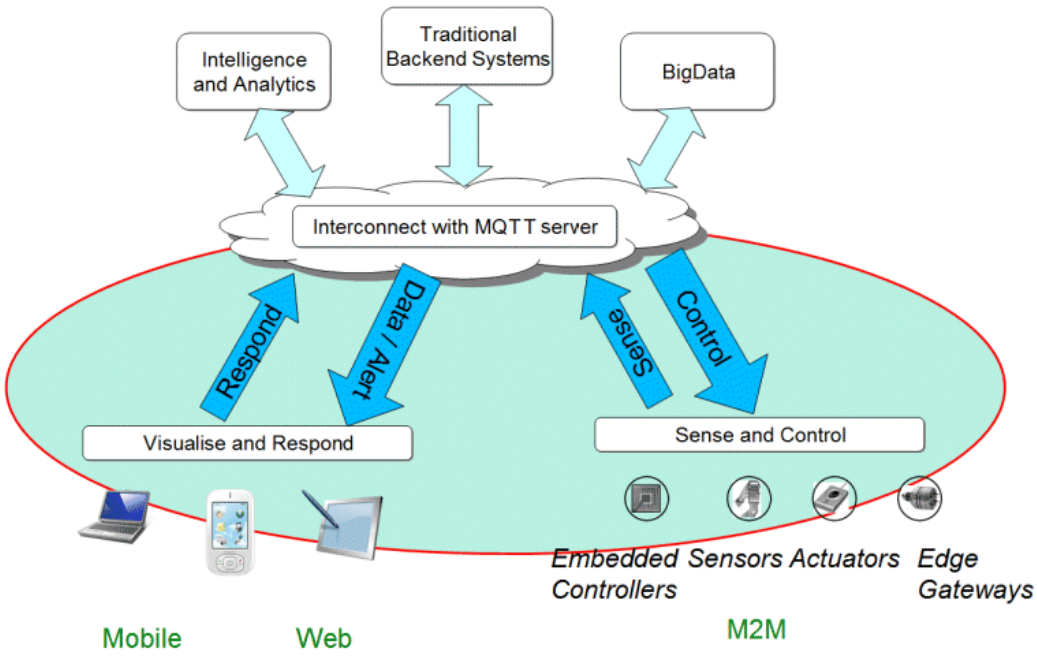

Figura 1.5: Schema logico funzionamento MQTT.

Un'alternativa a MQTT può essere *OPC-UA*[20] (OPC Unified Architecture) che è un protocollo industriale di comunicazione M2M gestito dalla *OPC Fundation* (OPC sta per Object Linking and Embedding for Process Control) ed è il successore di *Open Platform Communications* sviluppati nel corso degli anni da un consorzio di industrie che crea e mantiene degli standard di *open connectivity* per dispositivi e sistemi automatici industriali.

L'obiettivo principale di questa tesi risulta essere quello di sviluppare un sistema di gestione energetica che preveda, oltre che il controllo di energia proveniente da sistemi di *energy storaging*, *fotovoltaici* e da *energy provider* più classici, anche il controllo dei servizi aggiuntivi come quello anti-blackout nel caso venga superata la soglia massima elargita dal provider stesso. In Italia infatti, per ambienti domestici, è stato stabilito, come limite massimo di consumo erogabile dal provider, una quantità di energia pari a 3300 W. Può capitare molto spesso che, al superamento di questa soglia per più di un certo numero di minuti, venga bloccata l'erogazione di energia all'abitazione.

Il sistema proposto prevede la ricerca di qualcosa che vada a sopperire la rigidità di molti sistemi di automazione presenti nel mercato, ma che allo stesso tempo risulti essere non solo più economico, ma anche più dinamico ed adattabile in base alle esigenze strutturali dell'ambiente in cui lavora ed in base ai device che gestisce, e che si basi su tecnologie prettamente open-source.

Inoltre i sistemi di *home automation* creati con tecnologie libere ed open-source, ed implementati secondo le proprie necessità, possono usufruire di innumerevoli vantaggi sia dal punto di vista della *privacy* che della *security*, oltre che di customizzazione ed adattabilità. Infatti con la possibilità di avere sotto controllo ogni aspetto del sistema, sia dal punto di vista della progettazione hardware (seppur Raspberry Pi non sia completamente open-source, in quanto contiene un chip Broadcom con firmware proprietario) che da quella software, si ha una importante limitazione di tutte le problematiche riguardo la sicurezza e privacy che stanno affiorando con l'esplosione del mercato di dispositivi IoT e l'aumento dell'utilizzo di tali device, in moltissimi casi, closed-source.

Capitolo 2

Implementazione

Il sistema implementato provvederà quindi ad ottimizzare il consumo energetico domestico grazie a vari elementi funzionali interoperanti messi in comunicazione tra loro.
Da uno sguardo generale dall'alto, potremo descrivere il sistema come formato da elementi che forniscono energia all'ambiente domestico, altri che consumano energia, altri che calcolano e prendono decisioni e, altri ancora, che attuano le decisioni prese dalla precedente tipologia. Ovviamente anche questi ultimi due tipi di elementi sono da considerarsi come "consumatori" di energia, ma essendo device a basso consumo possono anche essere ritenuti come poco influenti al consumo energetico totale.
Tra gli elementi che "calcolano e prendono decisioni", vi è una componente hardware che merita un piccolo approfondimento in più. L'hardware in questione è il contatore ad impulsi. Un contatore ad impulsi è un contatore monofase di consumo energetico che permette di visualizzare tale consumo energetico in kWh. Il contatore, posto tra una fonte di energia ed un consumatore, emette un impulso ogni volta che una certa quantità di energia transita al suo interno. Normalmente è presente un display dove viene visualizzato il calcolo dell'energia transitata in kWh. In sostanza, genera impulsi in proporzione all'energia misurata. Nella fattispecie il contatore scelto emette 1000 impulsi ogni kWh misurato e grazie a questo calcolo si può misurare l'energia consumata istantaneamente.

A grandi linee il progetto può essere diviso in tre parti che provvederemo a spiegare in dettaglio più avanti.
Una che riguarda la simulazione dell'ambiente domestico, dei pannelli fotovoltaici e delle batterie. L'abitazione viene simulata all'interno di un PC, grazie ad un'interfaccia web.

Un'altra parte è quella del Raspberry Pi, ovvero il "cervello" che decide se e quando spegnere il carico interrompibile in base alle informazioni ricavate sia dalla casa (come detto in precedenza, il PC) che dai carichi interrompibili (o meglio, dagli Arduino collegati al carico interrompibile ed al contatore ad impulsi).
La terza parte è formata dai device Arduino che si interfacciano da una parte con il Raspberry Pi e dell'altra con il contatore ad impulsi ed il carico interrompibile vero e proprio, nel nostro caso una lampada con una lampadina da 100 Watt di potenza.
Queste componenti possono comunicare tra loro grazie ad un bus formato da cavi ethernet ed un router a formare una rete locale Ethernet.

2.1 Strumentazione

Come detto precedentemente, la realizzazione del progetto prevede l'uso di tre principali componenti. Un PC, un Raspberry Pi (modello B+) e due microcontroller Arduino con, in aggiunta, anche tutta una serie di strumenti fini al collegamento dei vari device, alla misurazione dell'energia e allo spegnimento dei carichi interrompibili.

2.1.1 Hardware

Di seguito una lista del materiale indispensabile:

Componente	**Descrizione**	**Quantità**
PC	Personal Computer da usare come simulatore dell'abitazione	1
Raspberry Pi	Model B+	1
Arduino Uno	Microcontroller	2
Router	Un router di collegamento tra device	1
Cavo Ethernet	Formano il bus, per collegare i device tra loro	4
Alimentatore Arduino UNO (9 o 12 Volt DC)	Per alimentare i device Arduino	2
Cavo Usb	Per caricare i device Arduino del codice necessario	1 o 2
Arduino UNO Shield	Board Shield per abilitare Arduino ad essere connesso ad una rete ethernet 100 Mbps	2
Relay Arduino	Relay per pilotare carichi fino a 5A come, ad es., una lampada da 100W.	1
Contatore ad impulsi	Contatore mono-fase di consumo energetico istantaneo dotato di uscita SO+/-	1
Cavetteria	Cavi per il cablaggio di potenza (220V AC), di colore nero (neutro) e Blu/Marrone (fase).	1
Nastro isolante	Nastro isolante per rifinire i cavi	1
Forbici da elettricista	Per tagliare e rifinire i cavi	1
Jumper Arduino M/F	Cavi di collegamento tra pin di device Arduino	Almeno 6
Presa di corrente	Presa di corrente per attaccare carichi, ad es. una lampada	1
Lampada	Lampade da usare come carico interrompibile.	1
Lampadina	Lampadina di potenza pari a 100 W	1

Tabella 2.1: Lista elementi hardware.

2.1.2 Software

Ogni device usufruisce di software e programmi open-source.
Nel caso della simulazione tramite PC:

- Archlinux
- HTML5
- JQuery
- PHP
- Apache

All'interno dei due device Arduino, il software utilizzato è quello standard di Arduino: Wiring [21].
Per quanto riguarda invece il Raspberry Pi abbiamo:

- Raspbian Jessie
- Python
- MySQL
- Cron

2.2 Schema Generale

Lo schema logico si basa sulla creazione di moduli software, che potremmo chiamare Adapter, con il fine di porsi come collegamento tra il bus ed i vari componenti software che simulano o si interfacciano con gli elementi funzionali interoperanti del progetto.

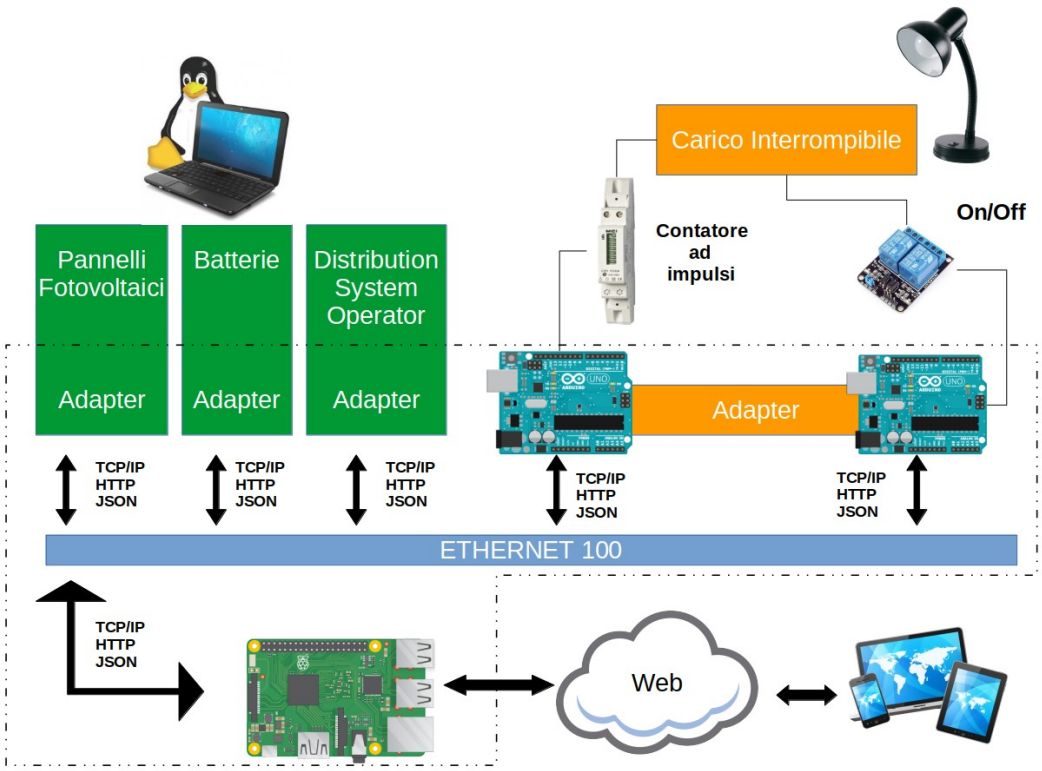

Figura 2.1: Schema logico-implementativo: in verde le parti simulate, in arancione quelle non simulate.

Dalla Figura 2.1 possiamo notare come è organizzato il sistema a livello logico-implementativo.
L'area tratteggiata rappresenta la parte software implementata.

2.2.1 Il simulatore

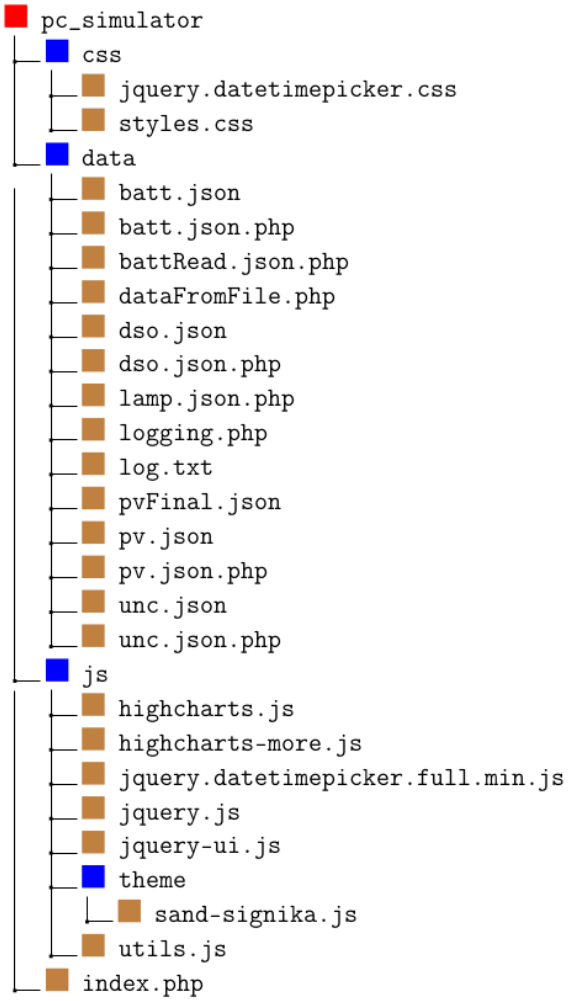

Figura 2.2: Struttura Simulatore.

I pannelli fotovoltaici, le batterie ed il Distribution System Operator (ovvero il provider che fornisce energia elettrica alla casa) sono stati simulati all'interno di un PC.
Il simulatore è un'interfaccia web composta da una parte front-end, formata da una pagina HTML5, con l'aggiunta di parti in PHP e Javascript, ed una parte beck-end che è un server Apache. Vediamo nel dettaglio.

Il lato front-end prevede un'organizzazione strutturale come in Figura 2.2. Al caricamento del file `index.php` viene visualizzata una dashboard composta

da vari indicatori simili a dei wattmetri che visualizzano l'energia misurata della batteria, delle celle fotovoltaiche, del fornitore di energia e del carico interrompibile. Inoltre, sotto i wattmetri, si presentano due grafici che mostrano, rispettivamente nel primo, il consumo dei vari componenti nell'ultima ora, mentre nel secondo, il consumo totale richiesto dall'ambiente domestico e la quantità di energia fornita all'abitazione (quindi sia dal DSO, che dalla batteria e le celle fotovoltaiche).

Dalla Figura 2.2 notiamo che la root è la directory `pc_simulator` ed al suo interno sono presenti tre sotto directory, `css`, `data` e `js`, ed un file `index.php`.
Il file `index.php` è il file da eseguire per accedere all'interfaccia del simulatore.
Per la creazione grafica dei grafici e dei wattmetri è stato utilizzata la libreria Highcharts [22] presente all'interno dei file `highcarts.js` e `highcarts-more.js` nella directory `js`.

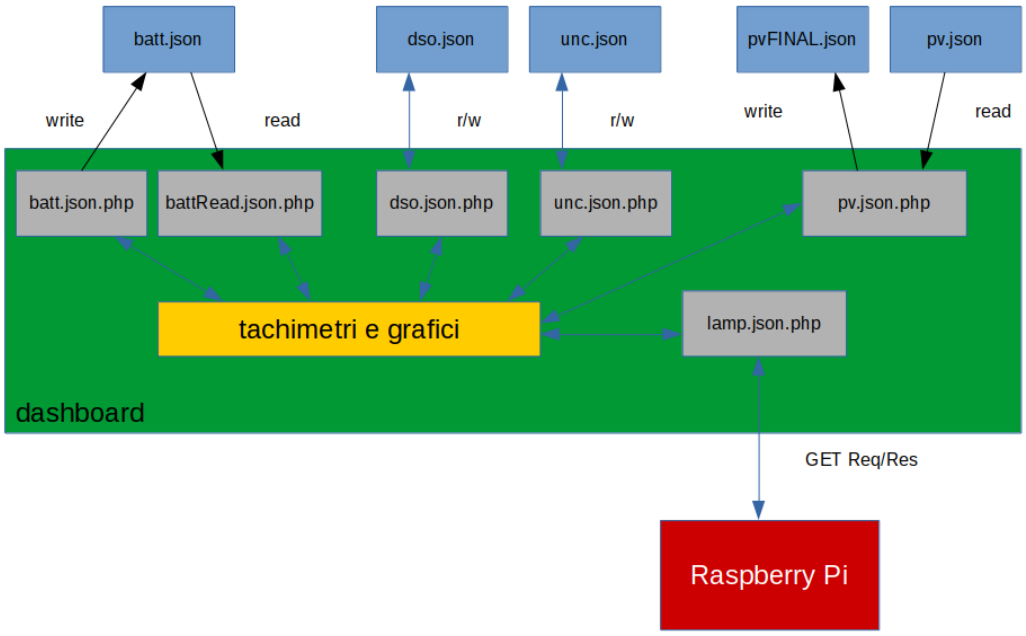

Figura 2.3: Schema logico del funzionamento dei moduli del simulatore.

I wattmetri (e di conseguenza anche i grafici) ottengono i dati grazie all'esecuzione di vari file `php` all'interno della directory `data`. Ogni minuto vengono aggiornati tutti i grafici presenti nella dashboard. Questi file `php`

leggono il valore della potenza dei compontenti presente all'interno di alcuni file `JSON`. Come vedremo nel proseguimento della descrizione del progetto, la struttura `JSON` è stata utilizzata sia come una sorta di database, sia come struttura utile alla comunicazione dei dati tra i device.
I `JSON` presenti all'interno della directory `data` (`batt.json`, `dso.json`, `pv.json` e `unc.json`) sono file `JSON` con formati da una coppia chiave/valore rappresentata da:

```
{"power":100}
```

ovvero la chiave `power` con il proprio valore dei Watt di potenza del componente.

La dashboard prevede anche una compontente di logging degli eventi. Il file `logging.php` in `data` fornisce la classe che aggiunge all'interno di un file di log, in questo caso `logging.txt`, una stringa di testo. `logging.php` viene chiamato dal Raspberry Pi per notificare particolari eventi come l'accensione o lo spegnimento del carico. Il file di log viene poi letto attraverso del codice `javascript` locato all'interno del file `utils.js` e caricato in una `textarea` visualizzabile cliccando nel apposito link in alto a destra della dashboard.

Il wattmetro della batteria legge il valore della batteria (che ricordo viene simulata) dal file `batt.json` grazie al file `battRead.json.php`. Invece il codice del file `batt.json.php` scrive (e quindi modifica) un nuovo valore della potenza all'interno del file `batt.json`. Questo perché deve essere prevista anche una variabilità del valore dell'accumulo elettrico nel tempo, dovuta all'utilizzo.

I dati del massimo valore erogato dal DSO vengono forniti al wattmetro attraverso il file `dso.json.php` che ricava il valore della potenza dal file `dso.json`. Il valore è inseribile nel campo sopra il wattmetro.

I dati dei carichi non controllabili (o non interrompibili) sono invece forniti al wattmetro grazie ai file `unc.json.php` ("unc" sta per uncontrolled) e `unc.json`, nella stessa maniera del caso del DSO. Anche se il valore del carico non controllabile può essere inserito nell'apposito campo d'inserimento.

Nel caso della visualizzazione del wattmetro del fotovoltaico, nel file `pv.json` sono salvati dei valori fissi di potenza media di 30 giorni, per ogni mese dell'anno, nell'arco di 24 ore. Più precisamente, un intervallo orario con valori positivi di potenza, con una frequenza di 15 minuti, per un totale di 58 valori di potenza per ogni mese. I dati sono stati estrapolati dal sito

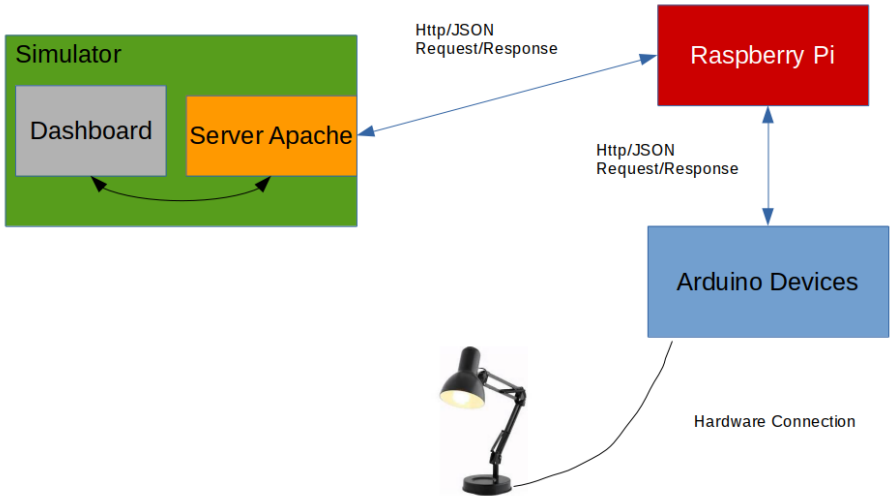

Figura 2.4: Schema della comunicazione tra componenti.

dell'Istituto dell'Energia e dei Trasporti della Commissione Europea [23]. Il file `pv.json.php` legge i dati presenti nel file `JSON` su richiesta di due menù a tendina posti sopra il contatore, dove si può scegliere il mese e l'orario preferito. Dal punto di vista grafico, il menù viene costruito dalla libreria `datatimepicker.full.min.js`[24].
Una volta estratto il valore, `pv.json.php` scrive nel file `pvFINAL.json` il valore definitivo di potenza del pannello fotovoltaico, in modo tale che la libreria grafica del wattmetro possa far visualizzare il valore nel wattmetro.

Per quanto riguarda il carico interrompibile (la lampada), il wattmetro chiama lo script `lamp.json.php` che effettua una richiesta `GET` al Raspberry Pi che provvederà a ritornare un `JSON` con al suo interno la classica coppia potenza/valore che rappresenta l'attuale valore della potenza del carico calcolata dal Raspberry Pi.

Il file `dataFromFile.php` è invece dedito alla lettura dei vari file `JSON` per poi ritornare il valore della potenza del componente richiesto. Verrà infatti chiamato dal Raspberry Pi per ricavare i valori delle potenze dei componenti simulati.
In Figura 2.4 è raffigurato uno schema della comunicazione tra i vari componenti descritti fin'ora.

2.2.2 RaspberryPi

Il Raspberry Pi rappresenta il cuore del sistema. Calcola la potenza stimata del carico interrompibile (secondo un algoritmo preciso che vedremo in dettaglio nella prossima sezione), decide se accendere o spegnere il carico, ed infine, sceglie cosa far scrivere nel file di log del simulatore situato nel PC.

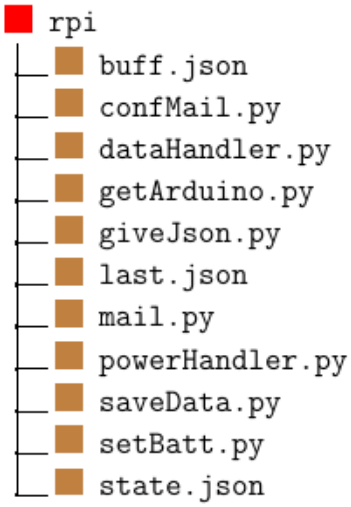

Figura 2.5: Struttura Rpi.

Dalla Figura 2.5 notiamo per prima cosa che questa parte del progetto si basa su codice scritto in `Python`. Tale scelta è dettata da mero gusto personale, più alcuni dettagli che hanno influenzato la scelta finale del linguaggio, come la facilità con cui si trovano librerie e moduli `Python` che implementano la comunicazione tra componenti attraverso protocolli Http classici oppure la gestione e la interazione con database MySQL. Inoltre `Python`, oltre che essere open-source, è anche molto ben integrato da sitemi operativi Linux e, nella fattispecie, Raspbian 8 (ovvero Debian Jessie per Raspberry).
I tre principali moduli del sistema sono `saveData.py`, `dataHandler.py` e `powerHandler.py`.

`saveData.py` è lo script che si incarica di effettuare, attraverso l'utilizzo del modulo `getArduino.py`, una richiesta GET al device Arduino collegato con il contatore ad impulsi. Il device Arduino risponde con una struttura `JSON` ed il modulo `getArduino.py` estrae il valore dell'impulso (conteggiato dalla board Arduino) e lo ritorna a `saveData.py`. Quest'ultimo salva il valore dell'impulso nel database. Se, per vari motivi (ad esempio, problemi

di connessione con Arduino), il valore ricevuto dalla board non è un intero (e quindi può essere la stringa della descrizione dell'errore ricevuto dalla chiamata GET ritornata da `getArduino.py`) allora viene salvato nel database il valore NULL e viene spedita una mail di avviso utilizzando il modulo `mail.py` che, con l'apposito file di configurazione `confMail.py` dove sono presenti username e password dell'account mail del mittente, sfruttando il modulo `smtplib` di `Python`, notifica alla mail desiderata il problema, inserendo nel testo della mail l'errore ricevuto nella response GET.
Dato che è stato introdotta una componente importante del sistema, il database, provvediamo a descriverlo in dettaglio.

Il Database

Il sistema di gestione del database scelto è stato, come accennato precedentemente, `MySQL`. Al database è stato assegnato il nome `power` ed è stato organizzato con due tabelle: `data` e `power`.

La tabella `data` è la tabella che si occupa di immagazzinare i dati "grezzi" provenienti dal dispositivo Arduino. In particolare, viene salvato in ogni record il valore dell'impulso ricavato dalla board Arduino con il proprio orario `timestamp`. Infatti la tabella `data` è formata da quattro colonne: `id`, `sensor_id`, `value` e `tstamp`.
La colonna `id` rappresenta il numero della entry ed è auto-incrementale, mentre `sensor_id` descrive il device Arduino (in previsione di avere più di un carico interrompibile). Il valore della colonna `tstamp` (di tipo `timestamp`) viene inserito automaticamente dal DBMS ogni volta che viene aggiunta una entry grazie agli attributi `CURRENT_TIMESTAMP` e `auto_increment`.
Viene utilizzata un'apposita tabella per i dati provenienti dai carichi perché il sistema utilizza tali dati all'interno dell'algoritmo per poi ricavare successivamente la potenza stimata del carico.

La tabella `power` è molto simile alla precedente se non per il fatto che il valore salvato al posto di `value` è `power` e rappresenta il valore *finale* della potenza stimata derivato dal calcolo dell'algoritmo computato dallo script `dataHandler.py`.

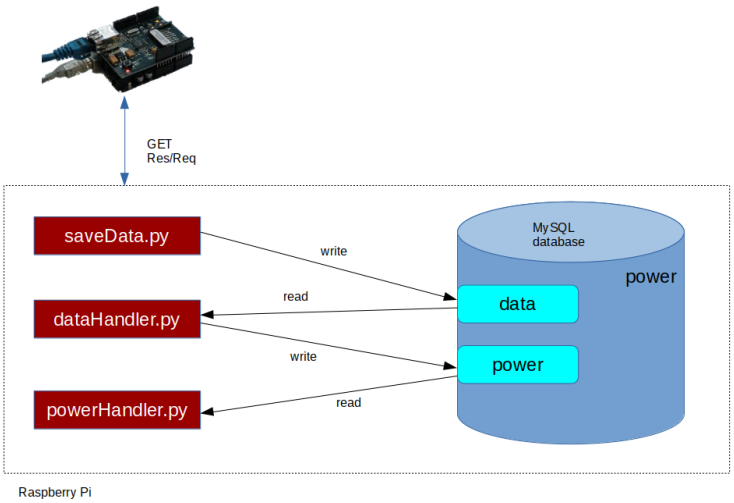

Figura 2.6: Interazione tra moduli e database

Lo script `dataHandler.py` ha il compito di calcolare il valore finale della potenza del carico. Una volta ricavato il valore dell'impulso dalla tabella `data` del database, se il valore è un intero allora calcola la potenza e salva il valore finale stimato nella tabella `power`, altrimenti sempre nella tabella `power` salva la entry con, nel campo `value`, il valore `NULL`. Ovviamente, in entrambi i casi, ricava il valore del `tstamp` dalla tabella `data` e provvede ad inserirlo nell'appropriata entry della tabella `power`. L'algoritmo, come vedremo, fa uso di alcuni valori temporanei utili alla riuscita del calcolo della stima. Questi valori sono salvati in `buff.json`. Inoltre `dataHandler.py` ha il compito di aggiornare un valore temporaneo, utile allo script `powerHandler.py`, situato in `last.json` che fornisce il valore più alto, fin'ora calcolato, della stima finale della potenza.
Il Raspberry Pi si occupa anche di fornire i valori della potenza stimata al simulatore. `giveJson.py` ha infatti la funzione di fornire un Python Web Server attraverso il framework Bottle [25], riceve quindi richieste GET alla porta 8080 e risponde con un `JSON` con all'interno il valore della potenza ricavata dal database.

Studio e progettazione dell'algoritmo

Potenza e Contatore ad Impulsi

Uno dei problemi principali che sono stati riscontrati durante l'implementazione del progetto è stato quello di trovare una soluzione soddisfacente ai limiti di misurazione che un contatore ad impulsi comporta.
Il sistema infatti si basa, per grandissima parte, sulla stima della potenza del carico interrompibile. Dato che il progetto prevede una misurazione quasi istantanea di tale potenza, il fatto che un contatore ad impulsi non estragga un valore reale della potenza ma, come dice il nome stesso, conti gli impulsi emessi in base al flusso di energia elettrica transitata in esso, ha richiesto di cambiare approccio all'idea di "misurazione istantanea" e di "stima finale della potenza".
La soluzione infatti più corretta è risultata quella di fornire al sistema una misurazione sicuramente non accurata al 100% ma una stima molto vicina al valore reale, ed inoltre il calcolo, come vedremo, viene "aggiustato" nel tempo attraverso un assottigliamento dell'errore. Quest'ultima parte è dovuta proprio alla *problematica intrinseca di un contatore ad impulsi nel misurare una potenza istantanea.* Il contatore infatti emette un tot di impulsi per kWh, nel nostro caso 1000 Imp/kWh ovvero un 1 impulso per Wh. Quindi se in un'ora vi fosse una variazione di impulsi pari ad uno, vorrebbe dire che è stata consumata una quantità di energia pari ad 1 Wh.
Il nostro scopo è però quello di utilizzare il contatore per fornire una misurazione di potenza (lavoro al secondo) quasi istantanea al sistema, e non una misurazione media della potenza in un'ora. Perciò si è pensato di misurare la differenza degli impulsi contati dal contatore (ed interposti dalla board Arduino) in un minuto e moltiplicare questa cifra per 60, che rappresenta la potenza media richiesta in tale minuto, calcolata in W. Per esempio, se avessimo al minuto 1 un valore di impulsi pari a 20 e al minuto 2 un valore pari a 22, avremmo:

```
(22 - 20) * 60 = 120
```

Il valore ottenuto, `120`, risulta essere i Watt di potenza calcolati in quest'ultimo minuto. Ovviamente il limite del calcolo della potenza consiste proprio in questo: la misurazione viene fatta *ogni minuto* e viene calcolata *confrontando il valore attuale con quello precedentemente ottenuto.* Attenzione però: si potrebbe calcolare anche in ogni secondo (e quindi moltiplicando per 3600), ciò non toglie che ci si dovrebbe rapportare con il concetto di

differenza di impulsi e soprattutto con il fatto che per consumi di Watt piccoli non si avrebbero differenze di impulsi nell'arco di un buon numero di secondi. La scelta di calcolare la potenza ogni minuto infatti è maturata durante la progettazione. Si è notato che per carichi alti (ad esempio 1000 W) la misurazione per secondo poteva essere una buona scelta. Ma per carichi bassi (come la nostra lampadina da 100 W) sarebbe stato uno spreco di risorse computazionali e di banda inutile, proprio dovuto al fatto che per "misurare" la potenza di un carico piccolo (o meglio, far variare il numero degli impulsi del contatore, collegato ad un carico piccolo) si necessita di un maggiore numero di secondi.
Chiarita questa prima parte, ci si è imbattuti però in un'altra difficoltà non meno importante. Il fatto di calcolare un valore nuovo utilizzando, nel calcolo, una differenza tra due valori di cui uno precedente, ha fatto riscontrare un problema nell'accuratezza del valore calcolato nei casi in cui, o non ci siano valori precedenti, o quando questi siano pari a 0 (per esempio all'avvio del sistema o di Arduino o Raspberry Pi, o quando il carico risulta essere spento). In tale circostanza infatti si avrà un valore che risulta essere discordante dal risultato atteso (magari risulterà essere sovrastimato) proprio perché è stato misurato nell'arco di un minuto e la variazione di impulsi comporta una misurazione di un *diverso* e *nuovo* valore esattamente al momento della stessa variazione del valore degli impulsi e non prima. Ma nel caso in cui il valore degli impulsi rimanga lo stesso, allora potrebbe voler dire che il carico potrebbe essere spento perché comunque sia, oltre certi livelli di carico (che specificheremo più avanti), se il carico risulta essere acceso, allora nell'arco di un minuto ci sarà sempre una variazione del numero degli impulsi.

Aggiustamento del valore nell'arco del tempo

L'efficienza dell'algoritmo implementato è riposta nel fatto di riuscire ad adattarsi a potenze di carichi diversi ed a correggersi adeguatamente nell'arco del tempo. Questo è dovuto al fatto che dopo vari studi è stato scelto un algoritmo che impone l'utilizzo di un valore tampone che permette tale correzione del valore finale.
Infatti vengono sommati, all'ultimo valore temporaneo, tutti i valori della stima (calcolata con la differenza tra gli ultimi due valori e poi moltiplicata per 60) precedenti fino all'ultimo maggiore di zero più lontano nel tempo ed il risultato viene diviso per il loro numero. In pratica viene fatta una "media mobile" di tutti i precedenti valori fino all'ultimo valore positivo più lontano dall'attuale. Il risultato ottenuto risulta essere il valore **finale** della stima della potenza. Questi valori temporanei vengono salvati in un file denominato `buff.json`, di volta in volta, ogni minuto. Chiaramente al primo

valore della tabella `data` pari a 0 o `NULL` i valori temporanei vengono azzerati.

Algorithm 1 Pseudocodice Algoritmo

```
estimate ← readFromJson(buf[estimate])
past ← readFromJson(buf[past])
time ← timeOfLastValueFromDB()
List pulses ← last5MinValFromDB()
if pulses[0] = NULL then
    FINAL ← NULL
    addToDB(FINAL,time)
    exit(0)
end if
if pulses[1] = 0 AND pulses[0] = 0 then
    power ← 0
else
    if abs(pulses[1]-pulses[0]) != 0 then
        power ← abs(pulses[1]-pulses[0])*60
    else
        power ← 0
    end if
end if
if estimate = 0 then
    FINAL ← power
    newPast ← 1
else if power = 0 then
    FINAL ← power
    newPast ← 0
else
    FINAL ← ((power+estimate)/(past+1))
    newPast ← past+1
end if
if power = 0 then
    newEstimate ← 0
else
    newEstimate ← estimate+power
end if
writeToJson(buff.json, newEstimate, newPast)
addToDB(FINAL,time)
```

Dall'algoritmo descritto in precedenza in pseudocodice possiamo osservare che vengono utilizzati dei valori tampone (riga 1 e 2) per valori precedentemente calcolati. `estimate` è la chiave del valore pari alla somma totale dei precedenti valori, mentre `past` è il numero dei precedenti valori sommati in `estimate`.
Dalla riga 4 alla 8 vengono prima presi i valori degli impulsi dalla tabella `data` del database relativi agli utlimi 5 minuti se ci sono, altrimenti se l'ultimo valore ricavato è `NULL`, viene inserito come valore finale nella tabella `power` il valore `NULL`.
In caso i valori siano diversi da `NULL` (dalla riga 10 alla 18), allora vengono controllati gli ultimi due valori e se sono entrambi pari a 0, viene assegnato ad un valore temporaneo della potenza il valore 0. Altrimenti viene fatta la differenza tra i due e, se diversa da 0, il proprio valore assoluto viene moltiplicato per il fattore precedentemente nominato di 60 (i secondi) ed il ricavato viene assegnato al valore temporaneo di potenza. Se la differenza è pari a zero allora il valore temporaneo viene settato a 0.
Nelle righe restanti, se il valore temporaneo della potenza è pari a 0 o il valore tampone con chiave `estimate` del file `buff.json` è pari a 0 allora il valore finale della potenza sarà 0. Altrimenti viene sommato il nuovo valore temporaneo della potenza alla somma dei precedenti valori ed aggiornato il numero di tali valori in `past`.
Infine, nelle ultime due righe, vengono aggiornati nel file `buff.json` i file temporanei e viene salvato il valore *finale* della potenza nella tabella `power` del database con il proprio `timestamp`.

Il file `powerHandler.py` ha il compito di comunicare con il simulatore e, soprattutto, di gestire l'accensione e/o lo spegnimento del carico.
Ricava i valori della batteria, del DSO, della cella fotovoltaica e del carico non interrompibile effettuando chiamate al file `dataFromFile.php` visto in precedenza nel Simulatore, responsabile della lettura dei valori dai vari file `JSON` contenenti il valore richiesto delle potenze dei componenti. Utilizza anche lui dei file per salvare vari valori importanti per la riuscita della gestione del carico.
Per prima cosa, una volta ricavati i valori delle potenze dei componenti, controlla se i pannelli fotovoltaici sono in grado di erogare un valore di energia elettrica significativo. Se questo avviene, "accende" il carico ed invia un messaggio di log al Simulatore. Da approfondire il fatto che se il carico risulta essere già acceso, accende lo stesso il carico, o meglio, invia una richiesta GET alla board Arduino che, collegata al Relay Arduino, accende il carico "chiudendo" il Relay collegato fisicamente al filo di fase della spina della corrente del carico.
Se i pannelli fotovoltaici non sono in grado di erogare potenza, allora controlla

se c'è abbastanza energia immagazzinata nella batteria. Se c'è, provvede alla simulazione del consumo della batteria diminuendo il valore della batteria stessa. Il calcolo viene effettuato sottraendo al valore attuale della potenza immagazzinata dalla batteria il valore richiesto dal carico, diviso per 60. Ovvero, dato che la batteria immagazzina potenza per ora (kWh, e quindi energia), allora viene tolto, ogni minuto (dato che calcoliamo ed aggiorniamo i valori ogni minuto), il valore richiesto diviso 60 secondi; ciò viene eseguito dal file `setBatt.py`. Viene anche segnalato il fatto di loggare l'evento, anche in questa circostanza, se lo stato precedente del carico fosse stato spento. In generale, il file `state.json` viene utilizzato proprio per segnalare, al prossimo controllo, se lo stato attuale del carico sia acceso o spento, per far sì che venga loggato l'evento "nuovo" e non ogni volta che si invia un messaggio di accensione o spegnimento al carico in questione.
Se la carica della batteria non risulta essere abbastanza, allora l'assorbimento della potenza del carico ricadrà nel DSO. Però viene prima controllato se il valore massimo del DSO (di solito, per un'abitazione domestica, pari a 3300 W) non venga superato dalla somma della potenza richiesta dal carico e della potenza del carico non interrompibile. Infatti se tale potenza viene superata, viene controllato se si può utilizzare una combinazione tra il DSO e le celle fotovoltaiche. Se non c'è abbastanza potenza, allora si controlla se ci sia con una combinazione tra batteria e DSO. Altrimenti si spegne il carico. Ed ovviamente si invia, se ritenuto corretto, il messaggio di log al Simulatore. Da notare il fatto che ogni volta che si decide di accendere il carico, viene controllato se poi, all'accensione stessa, non venga superato il valore massimo del DSO. Infatti in tal caso si creerebbe una condizione di accensione e spegnimento del carico ad intermittenza ogni minuto. Per fare ciò viene salvato all'interno del file `last.json` l'ultimo maggior valore di potenza fin'ora calcolato. In realtà, questo compito viene eseguito da `dataHandler.py`, e `powerHandler.py` legge e decide se accendere o no il carico, controllando se la somma dei valori dei carichi non interrompibili a quello salvato in `last.json` superi o no il valore soglia del DSO.
Una cosa importante da rimarcare è quella della scelta di caricare la batteria ogni qual volta che si abbia una eccedenza di potenza proveniente da qualunque altro componente che alimenti l'abitazione, in modo tale da ottimizzarne la gestione energetica.

2.2.3 Arduino

Per quanto riguarda la parte che si interfaccia direttamente con il carico interrompibile, ovvero le board Arduino, il linguaggio utilizzato è ovviamente quello standard di Arduino [21]. Dato che le board Arduino UNO utilizzate nel progetto sono due, una collegata al contatore ad impulsi ed un'altra collegata al relay Arduino collegato a sua volta al filo di fase della spina del carico interrompibile (potrebbe essere utilizzata una sola board per gestire sia il conteggio degli impulsi che la gestione dell'alimentazione del carico ma, per rendere ancora più reale la simulazione della gestione si è deciso di dedicare una board Arduino per ogni funzione), sono state compilate e caricate attraverso l'Ide fornito da Arduino stesso due versioni differenti dello stesso codice sorgente. Spieghiamo meglio.
La caratteristica del codice in questione è la modularità. Infatti, come si può notare analizzando il sorgente, all'inizio sono presenti molte `#define` incluse all'interno di condizionali di compilazione per il preprocessore come `#ifdef`. Ciò è stato fatto perché il codice prevede l'uso di molte parti che nel nostro caso specifico non sono utili ma che possono essere utilizzate in altre board per altri scopi o a contatto con altre implementazioni hardware. Il codice utilizzato, infatti, proviene dal raggruppamento di vari altri progetti che implementavano soluzioni diverse, per altri moduli harware di Arduino come, per esempio, le shield wireless, sensori termici, ambient light, più tipi di shield ethernet e tipi differenti di relay Arduino, e l'uso di una microSD per dati in configurazione ethernet o wireless. Inoltre, sono presenti anche tutte quelle parti che possono essere utili in molti progetti o che comunque sono molto comuni, come per esempio il serial debugging, la gestione della memoria, l'avvio del webserver o l'uso del watchdog per il reset della CPU.
Ma la particolarità risiede proprio nella possibilità di scegliere le varie parti a piacimento o in base alle proprio esigenze. Come possiamo vedere, all'inizio del codice, nel raggruppamento degli `#include` (tra questi è presente anche una libreria presa in prestito da `Github`, per gentile concessione di [26]) è presente anche l'inclusione di un file esterno denominato `config.h`. Questo file presenta al suo interno tutte le varie `#define` utili, se decommentate, alla "attivazione", all'interno del file che vogliamo caricare nella board, di quelle parti di codice presenti nei condizionali di compilazione. Infatti, nel caso della board per la scheda che si presta da interfaccia e contatore per il contatore ad impulsi (`ASU_Counter_Pulse.ino`), allora sono state decommentate: `#define BOARD_ETHERNET_SHIELD_W5100`, `#define FUNCTION_WEBSERVER`, `#define WATCHDOG` e `#define PULSE_SO` per abilitare rispettivamente lo shield ethernet, le funzionalità webserver, il watchdog di Arduino ed infine l'implementazione per il contatore degli impulsi prove-

nienti dal contatore stesso. Vediamo più in dettaglio queste funzioni.
La parte relativa all'implementazione dello shield ethernet W5100 si basa sull setting dei digital pin 4 e 10 in stato `HIGH`, la configurazione manuale del MAC address della board e del suo indirizzo IP. Infatti, come vedremo più avanti, ad ogni device interconnesso verrà assegnato un IP statico all'interno della LAN.
La presenza della macro `#define BOARD_ETHERNET_SHIELD_W5100` definisce una nuova macro `#define ETHERNET` che oltre ad attivare le funzioni che implementano lo shield ethernet, attiva anche la configurazione della porta 80 per il web server. Il web server però si avvia grazie alle funzioni di base di Arduino. Inoltre è stata implementata anche tutta la parte di risposta alla richiesta GET proveniente da Raspberry Pi. Le funzioni `handleClientRequestEthernet()` e `handleHttpRequest()` si occupano di rendere disponibile il server a richieste Http e poi rispondere di conseguenza. Nel caso si sia configurato il codice per la board in comunicazione con il Relay (ovvero `ASU_Relay.ino`), grazie alla `#define ACTUATOR_RELAY_SAINSMART_5V_2RELAYS` che fa settare i pin 2 e 3 per contenere le richieste per i 2 Relay, è stata imlementata anche una risposta consona alla richesta da stampare nel serial debug e come risposta alla richesta GET stessa. La risposta prevede l'invio di un `JSON` con all'interno informazioni base sull'IP del device ed il MAC address (dato che è stato in entrambi attivato il webserver ethernet), e in base all'attivazione delle funzioni si possono ricavare informazioni sulla memoria, sui vari sensori di temperatura, umidità e luce. Nel nostro caso inoltre sono state attivate la risposta alla richiesta di apertura o chiusura del Relay (quindi spegnimento o accensione del carico) e le informazioni riguardo al conteggio degli impulsi del contatore. Questa caratteristica è fornita grazie alle funzioni `checkPower()`, `onPulse()` e `pulseMeterSetup()` che permettono il "turnaround" del contatore interno della board Arduino, il conteggio elementare degli impulsi e l'attivazione del pin in ascolto dell'interrupt proveniente dal contatore stesso.
Il watchdog di Arduino implementato nel codice si rifà alla funzione `wdt_reset()` della libreria di Arduino.

2.2.4 Configurazioni di rete e configurazioni varie

La topologia presentata è proposta a solo titolo di esempio, è la topologia usata negli esperimenti. Il lavoro presentato in questa tesi si può applicare ad un ampio spettro di diverse configurazioni.
Il meccanismo di interconnessione del progetto è basato su request/response Http tra tutti i device presenti. Nel caso specifico, nel corso dello sviluppo del sistema, sono state riscontrate delle problematiche causate dalla posizione dei device all'interno dell'abitazione. Per esempio, il router utile per la comunicazione tra i device è risultato essere ubicato in una posizione scomoda nel poter collegare i device via cavo ethernet. La soluzione è stata dunque quella di usare due router: uno per collegare i device tra di loro via ethernet (e sostanzialmente anche rendere ancor più reale la simulazione, dato che nel progetto fin dall'inizio era prevista una rete ethernet di intercomunicazione tra le parti), ed un altro per poter far comunicare il Raspberry Pi (e volendo anche il PC) con l'esterno attraverso il wireless (questo perchè il Raspberry Pi potrà spedire email di avviso in caso di non comunicazione tra se stesso ed gli altri device).

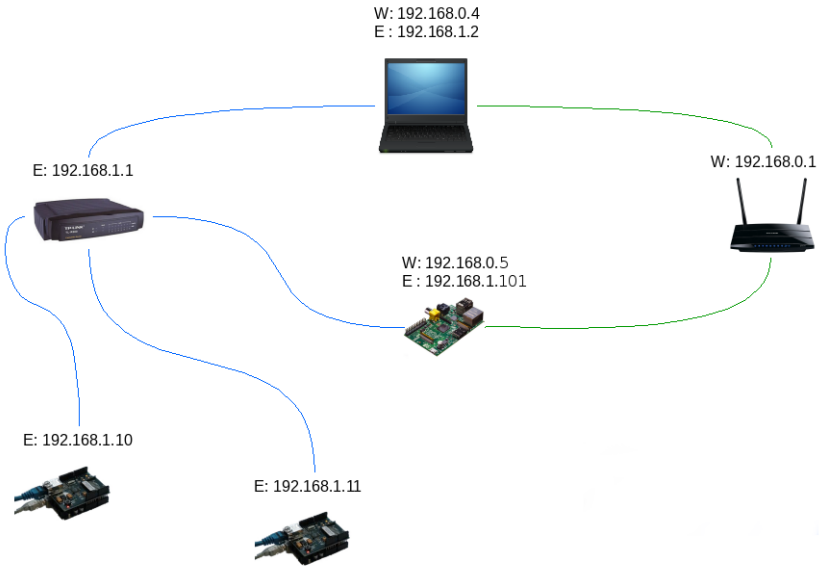

Figura 2.7: Configurazione Network: in verde connessione wireless, in blu connessione wired. W = interfaccia Wireless, E = interfaccia Ethernet.

Ovviamente per accuratezza bisogna far notare che il router collegato via ethernet è stato utilizzato come switch. Uno switch è il device più adatto

per questo tipo di funzione. Si è deciso di utilizzare un router solo perché comodità personali e per facilità di reperimento.
La Figura 2.7 mostra come è stata pensata l'organizzazione della rete del sistema. In pratica, i device che hanno due interfacce diverse, come il PC ed il Raspberry Pi, sono stati sfruttati appieno collegando quella ethernet ad un router non collegato alla rete (al web, al mondo esterno) ma utile soltanto alla riuscita del progetto (come snodo di interconnessione interno tra i device), quella wireless invece collegata al router che poi si interfaccia con il ISP ed il web. Sono stati usati quindi IP statici in tutti e due i router.
Sono state dunque necessarie delle impostazioni in più nel device Raspberry Pi e nel PC. La prima si basa sulla modifica del file di configurazione di rete `/etc/network/interfaces` in tal modo da disabilitare il server DHCP al device ed assegnare un IP statico all'interfaccia ethernet:

```
source-directory /etc/network/interfaces.d

auto lo
iface lo inet loopback

#iface eth0 inet manual
auto eth0
allow-hotplug eth0
iface eth0 inet static
address 192.168.1.101
netmask 255.255.255.0

allow-hotplug wlan0
iface wlan0 inet manual
wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
```

L'interfaccia wireless è stata lasciata al classico `wpa_supplicant` della distribuzione, in quanto sarà il router stesso ad assegnare staticamente l'IP al MAC address dell'interfaccia del device.

Nel PC (che ricordo ha come sistema operativo GNU/LInux basato sulla distribuzione Archlinux) è stato disabilitato il server DHCP per la scheda ethernet. Questo perché ovviamente serviva assegnare un IP statico per l'interfaccia, ma anche perché l'assegnare l'IP solo tramite MAC address nel router e mantenere il DHCP della scheda ethernet attivo, fa sì che i server DHCP delle due interfacce (wireless ed ethernet) si riconoscano come gateway predefinito a vicenda per tutto il traffico di rete del device.

I device Arduino, come detto in precedenza, sono stati collegati al router ethernet che poi si occupa ad assegnare l'IP statico tramite MAC address e come da impostazione stessa all'interno della configurazione dei device (vedi sezione precedente).

Capitolo 3

Scenari e Casi d'Uso

Il progetto prevede il utilizzo del sistema descritto precedentemente in scenari domestici. Per questo, data la presenza di un simulatore, si è scelto di simulare 4 diversi scenari che possono essere avvenire realmente nella quotidianità di un utente.

3.1 Scenario 1

Il primo scenario prevede che la batteria abbia una carica energetica pari a 200 Wh, il DSO imponga un limite di 3300 W, i carichi non interrompibili siano per un totale di 300 Watt e che la lampada sia spenta.

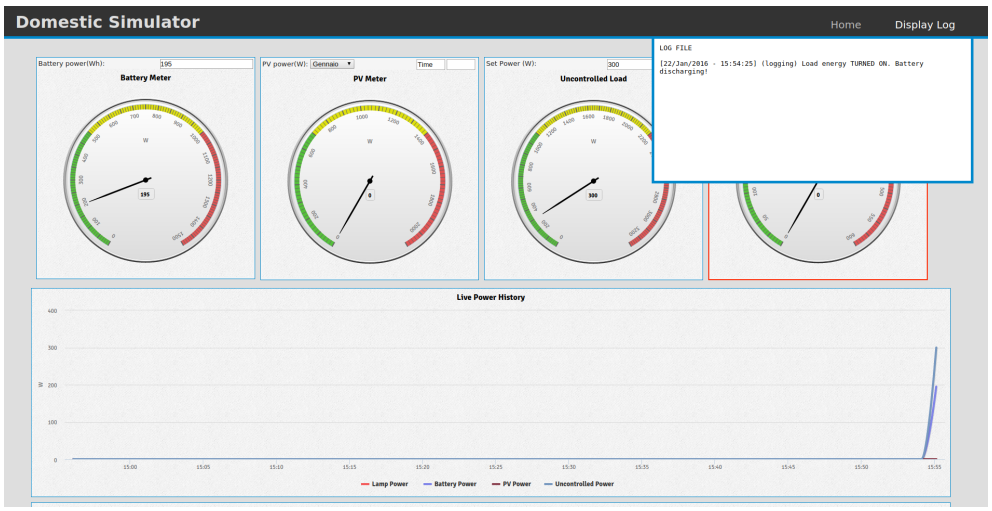

Figura 3.1: Scenario di scarico sulla batteria e notifica nel file di log.

Abbiamo appena detto che la lampada è spenta ma è da sottilineare il fatto che seppure la lampada per sé sia spenta, il carico interrompibile è *attivo*. Infatti la lampada risulta essere spenta perché l'utente non la ha ancora accesa, ma è comunque attivabile in quanto non è stato "aperto" il relay, è quindi bloccata l'alimentazione elettrica alla lampada.

In questa situazione, il totale consumo energetico è a carico della batteria, anche perché non abbiamo ancora selezionato le celle fotovoltaiche, e quindi tale scenario potrebbe essere riconducibile ad una circostanza attuabile alla mattina presto quando non è presente abbastanza luce per alimentare le celle fotovoltaiche e l'utente sta ancora dormendo.

Figura 3.2: Scenario di carico sulla batteria.

3.2 Scenario 2

Potremmo quindi simulare un ambiente in pieno giorno, con tempo soleggiato, dove il fotovoltaico sta erogando energia all'abitazione. Quindi abbiamo selezionato nel menù a tendina del wattmetro del fotovoltaico Gennaio alle ore 15:07. La lampada viene accesa dall'utente.

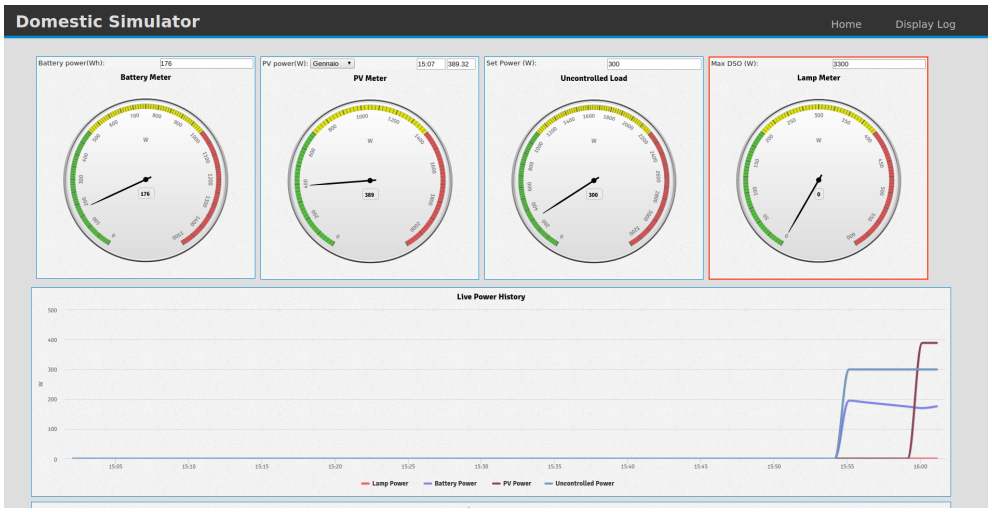

Figura 3.3: Scenario di carico sulle celle fotovoltaiche e caricamento della batteria.

In questo momento la batteria viene caricata dall'eccedenza di energia elettrica proveniente dalle celle fotovoltaiche.

3.3 Scenario 3

Nei precedenti scenari non vi è stata nessuna interruzione del carico. In questo terzo scenario, potremmo simulare una situazione nella quale l'utente usufruisce di vari carichi non interrompibili ed in più quello interrompibile. Potrebbe essere sera, non vi è irraggiamento solare e si stanno utilizzando vari elettrodomestici, con in più la lampada. Quindi aumentiamo, per esempio, il livello del carico non interrompibile da 300 a 3300 Watt, dall'apposito campo sopra il wattmetro. Cambiamo orario nel menù del fotovoltaico, inserendo così un valore basso o assente di energia elettrica proveniente dalle celle.

Figura 3.4: Scenario con consumo al limite dei carichi *non* interrompibili.

Avremo così una situazione nella quale la totale richiesta energetica ricade momentaneamente sulla batteria e, raggiunto il livello minimo di quest'ultima, viene spostata sul provider elettrico.

Figura 3.5: Scenario con interruzione del carico interrompibile.

Ma dato che si sta sforando il livello massimo di richiesta energetica allora avremo che il carico interrompibile viene spento, facendo in modo che la richiesta energetica dell'abitazione rimanga nei limiti impostati di 3300 Watt.

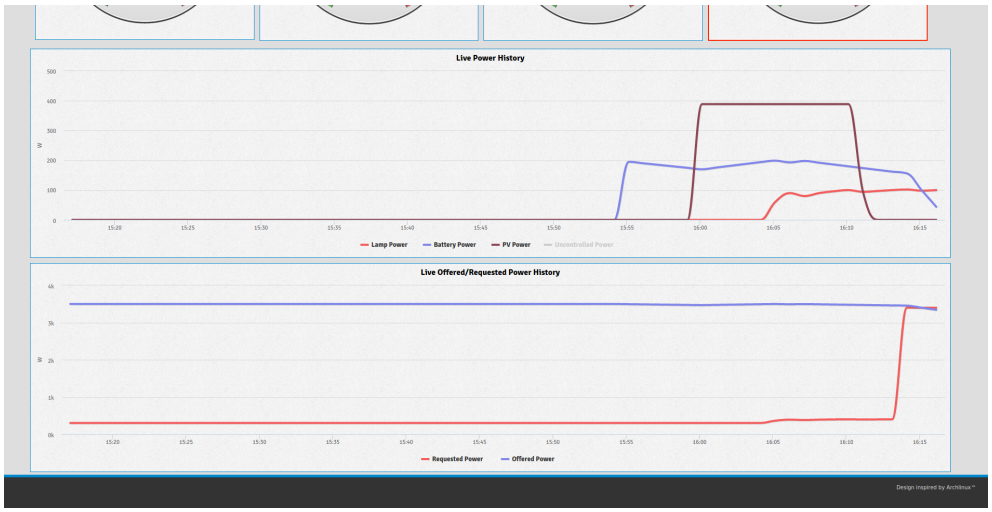

Figura 3.6: Scenario con interruzione del carico interrompibile, mostrando anche il grafico del consumo totale.

Nella Figura 3.5 e nella Figura 3.6 si può osservare che la linea che rappresenta i carichi interrompibili è stata deselezionata per rendere il grafico più leggibile.

3.4 Scenario 4

Un altro scenario si potrebbe basare sull'abbassamento del consumo del carico non interrompibile e di conseguenza si avrebbe che il carico interrompibile viene acceso, e quindi la lampada viene accesa.

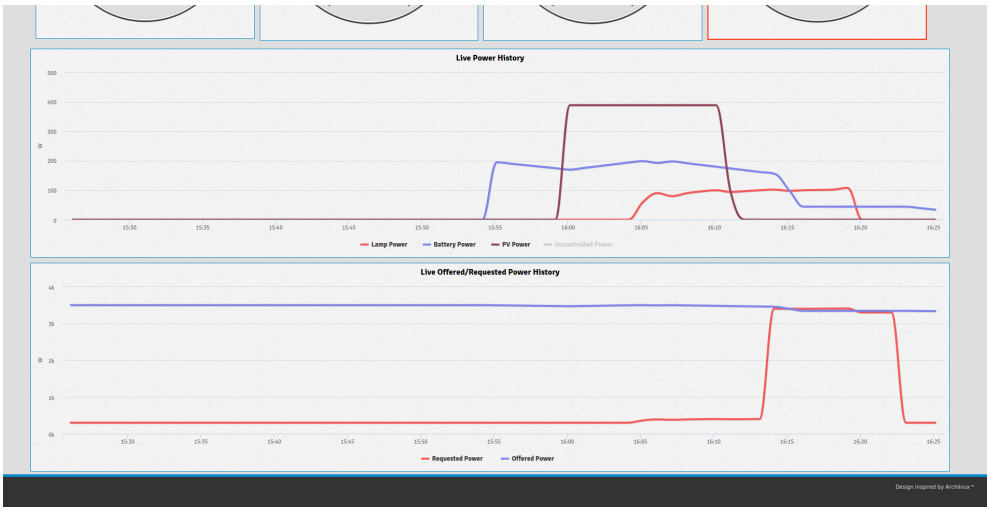

Figura 3.7: Scenario con livello dei consumi nei limiti e successiva accensione del carico interrompibile.

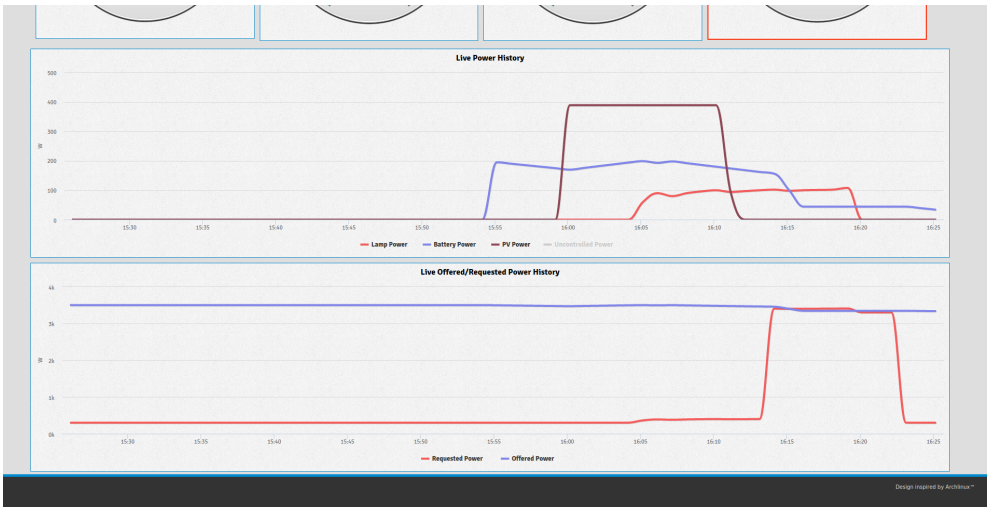

Figura 3.8: Altro esempio di scenario con livello dei consumi nei limiti e successiva accensione del carico interrompibile.

Tutte queste situazioni possono avere tante varianti. Da rimarcare è il fatto che il tutto viene notificato nel file di logging consultabile nel link in alto a destra e che è presente tutto lo storico del consumo energetico nei due grafici.

Capitolo 4

Evoluzioni future e conclusioni

Questo progetto è soltanto un primo passo verso un sistema molto più complesso di gestione energetica domestica.
Infatti possono essere aggiunti altri carichi interrompibili, ed il tutto è inseribile in ambienti reali e non simulati come in questo caso. La parte inerente alla gestione dei carichi, sia quella nel Raspberry Pi che quella nei device Arduino, può essere realmente utilizzata in circostanze reali in ambienti domestici.
Tale progetto è etichettabile come una versione 1.0 di un sistema ampliabile e rivedibile in una chiave più solida dal punto di vista della robustezza e della ottimizzazione del sistema.
Possono inoltre essere aggiunte funzionalità dal punto di vista della gestione dell'impianto stesso e della fruibilità delle informazioni attraverso un accesso remoto (con tutte le precauzioni del caso dal punto di vista della sicurezza contro exploit o danneggiamenti). Aggiungere statistiche del consumo giornaliero o in un qualsiasi intervallo temporale.

La versione in codice sorgente è presente nel repository pubblico
[https://github.com/cbarGit/HomeAutomatProject].
Il lavoro è stato rilasciato con licenza GPLv3.

Bibliografia

[1] Wikipedia. Energy demand management, . https://en.wikipedia.org/wiki/Energy_demand_management.

[2] Open EI. Demand side management. http://en.openei.org/wiki/Definition:Demand_Side_Management.

[3] Renewable energy policy network for the 21st century. Global status report. http://www.ren21.net/wp-content/uploads/2015/07/REN12-GSR2015_Onlinebook_low1.pdf.

[4] Bloomberg Energy Finance. Clean energy investment. http://www.bloomberg.com/company/clean-energy-investment/.

[5] IEA-pvps. Pvps report. a snapshot of global pv-1992-2014. http://www.iea-pvps.org/fileadmin/dam/public/report/technical/PVPS_report_-_A_Snapshot_of_Global_PV_-_1992-2014.pdf.

[6] GSE. Rapporto statistico sugli impianti fotovoltaici. http://www.gse.it/it/salastampa/GSE_Documenti/Il%20Solare%20fotovoltaico%202014.pdf.

[7] Confartigianato. Studio consumo energetico edifici residenziali, novembre 2015. http://www.confartigianato.it.

[8] Wikipedia. Internet of things, . https://en.wikipedia.org/wiki/Internet_of_Things.

[9] Nest Labs. Nest labs home automation. https://nest.com/.

[10] Honeywell. Honeywell. http://honeywell.com.

[11] KNX. Knx. http://www.knx.org.

[12] ZigBee Alliance. Zigbee. http://www.zigbee.org.

[13] OpenHAB. openhab.org. http://www.openhab.org.

[14] CalaOS. calaos.fr. https://calaos.fr/en/.

[15] Domoticz. domoticz.com. https://domoticz.com.

[16] Home Assistant. home-assistant.io. https://home-assistant.io/.

[17] Docker. docker.com. https://docker.com/.

[18] Open Motics. openmotics.com. https://openmotics.com.

[19] MQTT. Mqtt.org. http://www.mqtt.org/faq.

[20] Wikipedia. Opc unified architecture, . https://en.wikipedia.org/wiki/OPC_Unified_Architecture.

[21] Wikipedia. Wiring (development platform), . https://en.wikipedia.org/wiki/Wiring_(development_platform).

[22] Highcharts. http://www.highcharts.com/.

[23] Istitute of Energy and Transport. Commissione europea. http://re.jrc.ec.europa.eu/pvgis/.

[24] Valeriy xdan. Datatimepicker. https://github.com/xdan/datetimepicker.

[25] Marcel Hellkamp. Bottle web framework. http://bottlepy.org/docs/dev/index.html.

[26] maniacbug. Arduino memoryfree library. https://github.com/maniacbug/MemoryFree.

[27] Travis Oliphant. Numpy. http://www.numpy.org.

Allegati

Studio del calcolo della potenza di un carico interrompibile in ambiente domestico

Introduzione

Lo scopo è quello di studiare ed elaborare un algoritmo, il più possibile accurato e versatile, per la stima della potenza consumata da un carico interrompibile all'interno di un'abitazione domestica.
Il problema da risolvere consiste nel creare un buon algoritmo di stima basato su impostazioni precostituite che potrebbero limitarne l'accuratezza dei risultati e che allo stesso tempo sia anche versatile a livello di potenza reale dissipata dal carico interrompibile in questione.
Lo scenario nel quale l'algoritmo dovrà operare è raffigurato da un ambiente domestico con un carico che può essere acceso o spento. La misurazione della potenza è ricavata attraverso un contatore ad impulsi e avviene ogni minuto. Ogni minuto viene preso il valore degli impulsi contati e, attraverso un calcolo, misurata la potenza in quel minuto.
Si è perciò deciso di simulare una circostanza verosimile alla realtà, creando uno scenario di utilizzo del carico in questione da parte dell'utente. Lo scenario consiste nella rappresentazione, nell'arco di 120 minuti, di una serie di accensioni ed spegnimenti del carico fini alla simulazione di una situazione quotidiana, con lo scopo di verificare teoricamente ed a priori la correttezza degli algoritmi studiati, confrontando i risultati con la potenza reale dissipata dal carico. In sostanza, sono testati degli algoritmi di stima della potenza in un ambiente e scenario verosimile alla realtà, e vengono poi confrontati gli scostamenti e gli errori dei risultati, prodotti da ogni algoritmo, dalla potenza reale prestabilita. Viene inoltre studiata la differenza che ogni algoritmo produce per potenze diverse in un range che può essere presente all'interno di un'abitazione domestica. Infatti la quantità di energia elettrica che un provider può concedere ad un ambiente domestico è pari a 3300 Wh (Watt-ora, ed al superamento di tale soglia per oltre 3 minuti viene dismessa l'erogazione di

energia per ragioni di sicurezza e di contratto stabilito con il provider stesso). Quindi i vari algoritmi sono stati testati per ogni carico di potenza che varia da 1 a 3300 Watt.
Per creare lo scenario e gli algoritmi è stato utilizzato il linguaggio Python e librerie esterne molto conosciute in ambiente scientifico come NumPy.

Scenario

Lo scenario è rappresentato da 120 minuti di accensione e spegnimento del carico. Quindi, a livello astratto, tale situazione può essere rappresentata come un array nel quale ogni cella raffigura la situazione istantanea del minuto in cui viene calcolata la stima della potenza. Perciò ogni indice dell'array rappresenta un minuto da 0 a 120 minuti: l'array, dunque, è di lunghezza pari a 120, ed ogni cella può contenere due valori, 0 per la rappresentazione dello spegnimento del carico, e 1 per l'accensione.
La struttura necessaria alla raffigurazione del problema necessità però di altre informazioni, come la potenza reale, il valore della potenza calcolato dall'algoritmo, lo scostamento dalla potenza reale, etc..quindi un semplice array non può essere sufficiente. Si è deciso di utilizzare un array di array con lo scopo di rappresentare una sorta di tabella con ogni riga formata da un array, ed ogni array, attraverso le celle, che forma le colonne dove vengono rappresentati la situazione di accensione/spegnimento del carico, il valore degli impulsi calcolati in base alla potenza reale (quindi con virgola mobile), il valore degli impulsi virtualmente contati dal contatore ad impulsi (quindi il valore precedente senza virgola mobile), e poi i vari algoritmi di stima e le differenze con la potenza reale.
Il vantaggio di utilizzare una struttura del genere risiede nel fatto di poter aggiungere o rimuovere colonne a piacimento. Quindi, potenzialmente, possono essere aggiunte `n` colonne per `n` algoritmi di stima ed i propri scostamenti dal valore reale, aggiungere vari carichi e vari valori degli impulsi per contatori diversi, etc.
Il codice che crea il tutto (sia lo scenario che le varie stime) è all'interno di un unico file, `benchmark.py`. Analizziamo la prima parte.

```
#! /usr/bin/python

import json
import numpy as np
import matplotlib.pyplot as plt
import sys

#########################################################
############## data json input ##################

with open('params.json') as data_file:
    data = json.load(data_file)

#########################################################
############### data creation ###############

summPerc1 = 0
summPerc2 = 0
summPerc3 = 0
summPerc4 = 0
weight = 1
val = 0
rows = len(data.keys())

orig_stdout = sys.stdout
f = open('results.txt', 'a')###open file to write(append)
sys.stdout = f

########## import data from json #############

k = []
ki = []
k = list(data.keys())
for i in k:
        ki.append(int(i))
ki.sort()

for n,p in enumerate(range(1,3301)):
        power = p

        ########## create data arrays ###############
        matrix = np.ndarray((rows,12),dtype = object)
        matrix[:,0:] = int(0)
        matrix[:,2] = float(0)

        for key,a in zip(ki,matrix):
                a[0] = data[str(key)]

        #### power calculation ######
        for a in matrix:
                a[1] = a[0] * power


        ##### pulse calculation at seconds ##########
        for a,b in zip(matrix[1:],matrix):
                a[2] = float("{0:.3f}".format(b[2] + a[1]/60))

        ##### pulse calculation at minutes ##########
        for a,b in zip(matrix[1:],matrix):
                a[3] = int(b[2] + a[1]/60)
```

I moduli importati sono `json`, per l'utilizzo di strutture e file `JSON`, `sys` per leggere e scrivere su file, e `numpy`. Il modulo `numpy` [27] è un estensione del linguaggio `Python` che facilita la creazione di array multidimensionali e matrici, aggiungendo la possibilità di utilizzare un gran numero di funzioni matematiche ad essi associate, per la manipolazione dei dati. `NumPy` è un progetto open-source e *community driven*, ed è largamente utilizzato in ambito scientifico per i più svariati calcoli.

Il codice mostra che viene utilizzato un file `params.json`. Tale file contiene una struttura `JSON` dove vengono rappresentati in coppie chiave/valore il minuto e la situazione di accensione (1) o spegnimento (0) del carico. Da questo file, quindi, viene creato lo scenario.
La fase di pre-popolamento della "matrice" utilizza i dati appena caricati dal file per sapere quante righe (e quindi quanti minuti) devono essere create, e per formare delle liste Python che serviranno al popolamento della matrice stessa, precisamente al popolamento della colonna che riguarda lo stato di accensione/spegnimento del carico.
Con il `for` associato alla funzione `enumerate()` si apre il ciclo che permette di creare i 120 minuti di scenario per ogni potenza che varia da 1 a 3300 Watt. Vengono perciò creati 120 array per 3300 valori di potenza diversi.
Gli array stessi vengono creati con la funzione `ndarray` di `numpy`. Le righe saranno quante sono le chiavi del file `params.json` e le colonne sono 12, in quanto servono colonne per:

1. Numero del minuto
2. Valore Accensione o Spegnimento (1 o 0)
3. Numero Impulsi al secondo
4. Numero Impulsi al minuto
5. Stima Algoritmo 1
6. Errore Algoritmo 1
7. Stima Algoritmo 2
8. Errore Algoritmo 2
9. Stima Algoritmo 3
10. Errore Algoritmo 3
11. Stima Algoritmo 4

12. Errore Algoritmo 4

Dal codice precedente popoleremo la matrice solo fino alla quarta colonna; i dati per popolare le rimanenti colonne sono creati dagli algoritmi testati che, come è stato già accennato, sono quattro.

Studio e Testing degli algoritmi

Primo algoritmo di stima

Il primo algoritmo di stima è molto semplice. Per ogni minuto (quindi per ogni array con lo stesso valore di potenza reale) calcola la differenza tra gli impulsi al minuto del valore corrente con quello del precedente ed il risultato lo moltiplica per 60.

```
####################stima1#####################

        for a,b in zip(matrix[1:],matrix):
                if(a[0] == 1):
                        a[4] = (a[3] - b[3])*60

        for i in matrix:
                if(i[0]>0):
                        i[5] = abs(i[4]-i[1])
```

Poi riempe la quinta colonna con la differenza tra il valore di stima calcolato e la potenza reale.

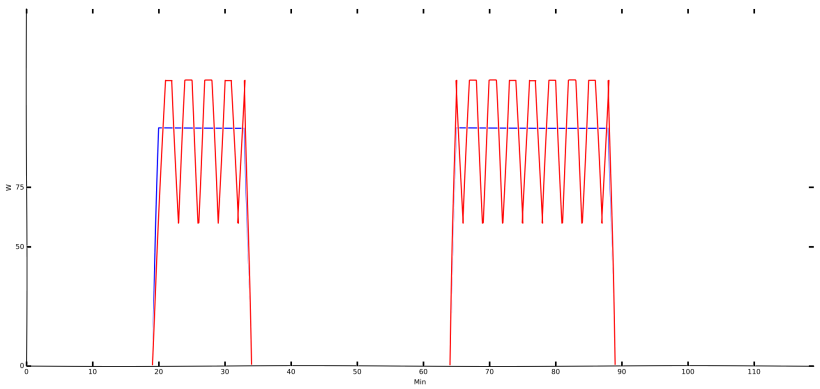

Figura 4.1: In blue la potenza reale (100 Watt) ed in rosso la stima dell'algoritmo 1

Dalla Figura 4.1 vediamo un esempio del risultato del primo algoritmo con un valore di potenza reale pari a 100 Watt.
Anche dall'immagine si evince che tale soluzione non può essere considerata soddisfacente, il valore stimato si discosta troppo dal valore reale ed inoltre non rimane neanche costante ma oscilla nel tempo.

Secondo algoritmo di stima

Il secondo algoritmo utilizza invece la differenza tra il valore degli impulsi al minuto corrente ed il valore degli impulsi calcolato 5 minuti prima, per poi dividere il risultato di tale differenza, moltiplicata per 60, per il numero di elementi presenti tra i due valori (compresi), quindi 6.

```
##############stima2##################

        pos = 6
        for i in range(0,rows):
                diff = 0
                if (i >= 6):
                        diff = matrix[i][3] - matrix[i-pos][3]
                        matrix[i][6] = diff*60/6

        for i in matrix:
                if(i[0]>0):
                        i[7] = abs(i[6]-i[1])
```

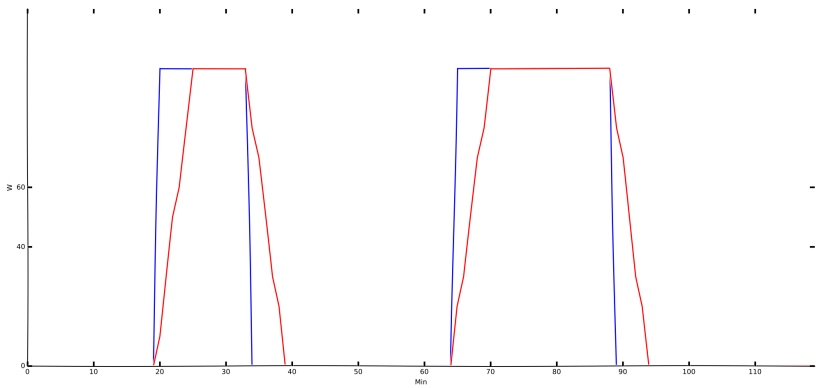

Figura 4.2: In blue la potenza reale (100 Watt) ed in rosso la stima dell'algoritmo 2

Nella Figura 4.2 si può notare subito che, seppur la stima calcolata risulta essere molto più accurata dell'algoritmo precedente, vi è un problema di velocità nel raggiungimento del valore calcolato che porterebbe quindi ad un peggioramento della reattività nella segnalazione di un carico troppo alto. Quindi anche in questo caso l'algoritmo deve essere scartato.

Terzo algoritmo di stima

Il terzo tentativo ha portato alla creazione di un algoritmo che applica il calcolo della potenza come nel primo algoritmo e aggiunge questo valore in un buffer. Poi somma tutti i valori presenti all'interno del buffer e li divide per il loro numero.

```
############ stima3 ###############

buff = []
for i in range(0, rows):
        summ = 0
        if( matrix[i][3] - matrix[i-1][3] != 0 ):
                if(i > 1 and (matrix[i][3] != 0)):
                        buff.append((matrix[i][3]
                                - matrix[i-1][3])*60)
                posi = list(filter(lambda x: x > 0, buff))
                summ = sum(posi)
                if(len(posi)!=0):
                        matrix[i][8] = summ/len(posi)
        else:
                buff = []
                        matrix[i][8] = 0

for i in matrix:
        if(i[0]>0):
                i[9] = abs(i[8]-i[1])
```

Valgono solo i valori positivi, quindi si applica una sorta di media tra tutti gli ultimi valori di potenza calcolati (con il primo algoritmo) fino all'ultimo valore precedente (a ritroso quindi) maggiore di zero. Il valore ottenuto è la stima della potenza per quel minuto.

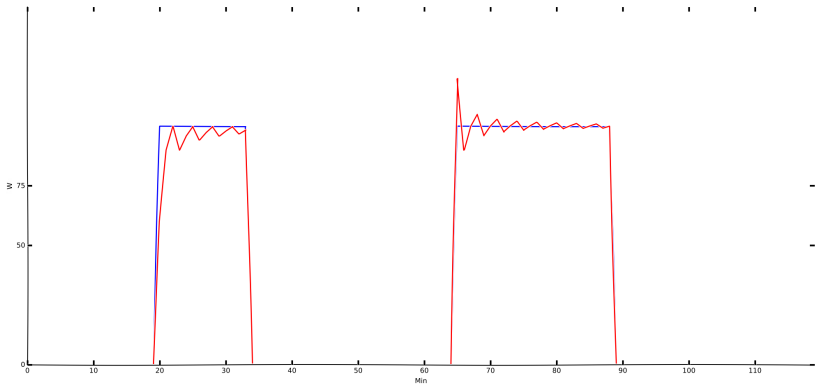

Figura 4.3: In blue la potenza reale (100 Watt) ed in rosso la stima dell'algoritmo 3

La Figura 4.3 mostra che questo algoritmo raggiunge un buon compromesso tra accuratezza e reattività nel calcolo del valore. Inoltre è anche abbastanza costante nel mantenere il valore.

Quarto algoritmo di stima

L'algoritmo n° 4 si comporta come il terzo, ma controlla che ci sia una variazione d'impulsi. Se questa non è presente, allora potrebbe non voler dire che il carico sia spento ma che la potenza sia troppo bassa da poter aumentare gli impulsi. Quindi la stima 4 è come se "rallentasse" la stima 3.

```
########################## stima 4 ##################################

c = 1
temp = 0
flag = 0
max_counter = 1
for i in range(0,rows):
        if(matrix[i][3]!=0):

                if( matrix[i][3] != matrix[i-1][3] ):
                        c = 0
                        temp = matrix[i][8]
                        if(matrix[i-1][3] == 0):
                                flag = 1
                        else:
                                flag = 0
                                matrix[i][10] = temp/max_counter
```

```
                else:
                        if(flag == 1):
                                max_counter += 1
                        matrix[i][10] = temp/max_counter
                        c += 1
                        if(c> max_counter):
                                matrix[i][10] = 0

for i in matrix:
        if(i[0]>0):
                i[11] = abs(i[10]-i[1])
```

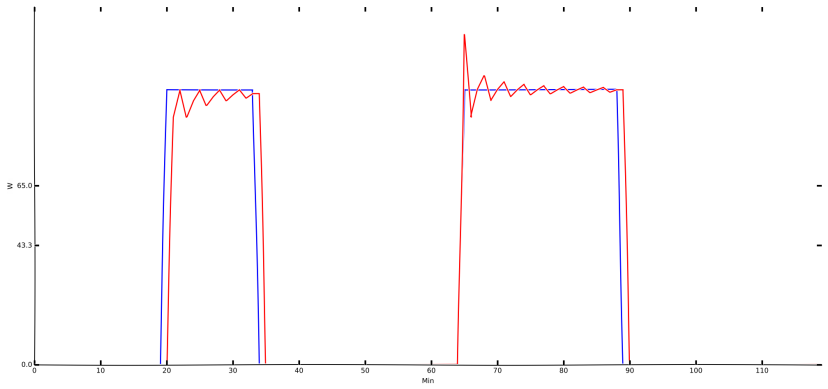

Figura 4.4: In blue la potenza reale (100 Watt) ed in rosso la stima dell'algoritmo 4

Come si può osservare nella Figura 4.4, l'algoritmo 4 risulta peggiorare la stima dell'algoritmo 3, in quanto si accorge in ritardo sia dell'accensione del carico che del suo spegnimento.

Scelta Finale

L'algoritmo migliore dal punto di vista dell'accuratezza e del rapidità nella risposta già dai risultati precedenti risulta essere il terzo. Ma il codice sottostante certifica ancora di più la correttezza della scelta.

```
############## final calculation ############################
        real = []
        stima1 = []
        stima2 = []
        stima3 = []
        stima4 = []

        for i in matrix:
                real.append(i[1])
                stima1.append(i[5])
                stima2.append(i[7])
                stima3.append(i[9])
                stima4.append(i[11])

        tot1=0
        tot2=0
        tot3=0
        tot4=0
        for i in stima1:
                tot1 += abs(i)
        for i in stima2:
                tot2 += abs(i)
        for i in stima3:
                tot3 += abs(i)
        for i in stima4:
                tot4 += abs(i)

        stima1Err = float("{0:.3f}".format(tot1/120))
        stima2Err = float("{0:.3f}".format(tot2/120))
        stima3Err = float("{0:.3f}".format(tot3/120))
        stima4Err = float("{0:.3f}".format(tot4/120))

        stima1ErrPerc =
                float("{0:.3f}".format((stima1Err/power)*100))
        stima2ErrPerc =
                float("{0:.3f}".format((stima2Err/power)*100))
        stima3ErrPerc =
                float("{0:.3f}".format((stima3Err/power)*100))
        stima4ErrPerc =
                float("{0:.3f}".format((stima4Err/power)*100))

        if (p <= 49):
                weight = 1

        if((p > 49) and (p % 100 == 0)):
                weight += 1

        avarageWeightErr1 = float("{0:.3f}".format(stima1ErrPerc
                * weight))
        avarageWeightErr2 = float("{0:.3f}".format(stima2ErrPerc
                * weight))
        avarageWeightErr3 = float("{0:.3f}".format(stima3ErrPerc
                * weight))
        avarageWeightErr4 = float("{0:.3f}".format(stima4ErrPerc
                * weight))
```

```
        if(avarageWeightErr1 > 100):
                avarageWeightErr1 = 0
        if(avarageWeightErr2 > 100):
                avarageWeightErr2 = 0
        if(avarageWeightErr3 > 100):
                avarageWeightErr3 = 0
        if(avarageWeightErr4 > 100):
                avarageWeightErr4 = 0

        summPerc1 += avarageWeightErr1
        summPerc2 += avarageWeightErr2
        summPerc3 += avarageWeightErr3
        summPerc4 += avarageWeightErr4

        ##########################################################
        ###### data output  ######################################

        print(n+1, stima1Err, stima2Err, stima3Err,
                stima4Err, stima1ErrPerc,stima2ErrPerc,
                stima3ErrPerc,stima4ErrPerc,'##',weight,
                '##', avarageWeightErr1,avarageWeightErr2,
                avarageWeightErr3,avarageWeightErr4)

        val += weight


#################################################################
################ decisor ########################################


print('The final values are:')
print(summPerc1/val,summPerc2/val,summPerc3/val,summPerc4/val)

ind = np.argmin([summPerc1/val,summPerc2/val,
        summPerc3/val, summPerc4/val])

print('...and the winner is')

if (ind == 0):
        print(' -> STIMA 1 <- ')
elif (ind == 1):
        print(' ->  STIMA 2  <- ')
elif (ind == 2):
        print(' -> STIMA 3 <- ')
else:
        print(' -> STIMA 4 <- ')

sys.stdout = orig_stdout
f.close()
```

Infatti per ogni valore di potenza reale compresa tra 1 e 3300 Watt, calcola le stime dei quattro algoritmi nello scenario prestabilito (120 minuti, con un certo numero di minuti di accensione e spegnimento). Per ogni stima, ricava l'errore medio in 120 minuti. Con questo valore calcola l'errore, confrontandolo con il valore reale, in percentuale, così da avere l'errore medio nei 120 minuti per algoritmo per ogni valore di potenza (compresa tra 1 e 3300 Watt). A questo valore in percentuale moltiplica un valore riconducibile ad

un "peso" che può avere il valore stesso di potenza reale. Perciò con valori minori di 50 Watt, il peso è pari a 1; da 50 Watt in poi ed ogni 100 Watt di valore, il peso aumenta di uno. Vengono quindi sommati per ogni stima i propri errori medi (nei 120 minuti) in percentuale e pesati con l'importanza della potenza reale. Infine avremo quindi 4 valori che verranno divisi per la somma dei valori "peso" precedentemente assegnate alle potenze.
Il risultato è:

```
The final values are:
0.4414203053462314 2.0999341029340988 0.12084567010492926
0.9166929791744616
...and the winner is
 -> STIMA 3 <-
```

Il tutto viene salvato in un file `results.txt`. Quindi è stato dimostrato che l'algoritmo migliore risulta essere il **terzo**.

Connessione hardware tra board Arduino UNO e contatore ad impulsi

Il contatore ad impulsi utilizzato nel progetto è un CONTAX 32A di MCI, raggiungibile al sito `www.mci-compteur-electrique.fr`.

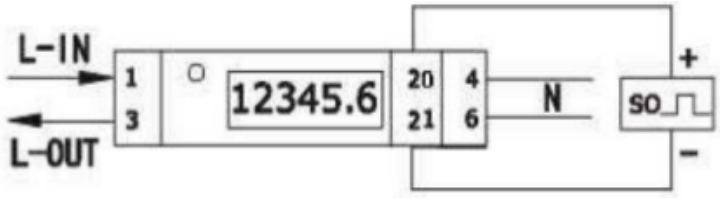

Figura 4.5: Schema di collegamento del contatore ad impulsi.

Per collegare il contatore al cavo elettrico in modo tale che il contatore stesso determini l'energia elettrica transitante in sé stesso, si deve tagliare il cavo elettrico del carico e separare il cavo di fase dal neutro. Una volta fatto ciò (e sguainato i cavi in modo tale da far esporre il cavo elettrico dalla guaina), si deve porre nel foro indicato con il numero 1, la linea entrante della fase, nel foro 3 la linea uscente della fase, mentre nei fori 4 e 6 rispettivamente le linee entrante e d uscente del cavo neutro come in Figura 4.6.

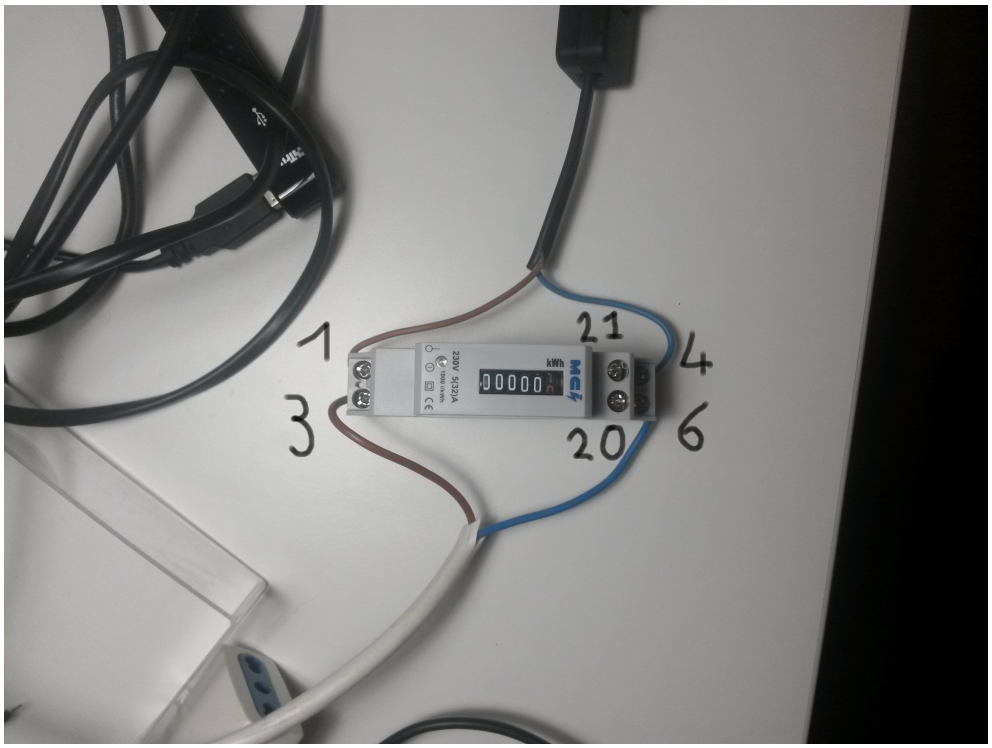

Figura 4.6: Collegamento del cavo elettrico con il contatore. In blu il cavo neutro, in marrone la fase.

Per collegare il contatore con la board Arduino UNO si necessita di una resistenza da 10 $k\Omega$ come nella Figura 4.7.

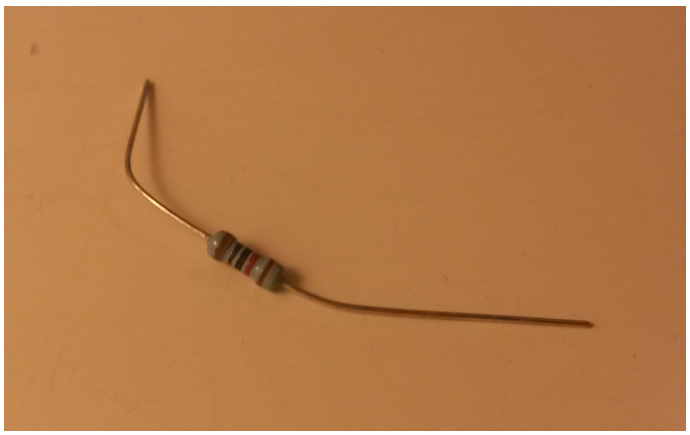

Figura 4.7: Resistenza da 10 $k\Omega$.

Bisogna quindi collegare il foro 21 al GND Arduino. I fori 20 e 21 sono i poli positivi e negativi dell'emettitore d'impulsi del contatore.

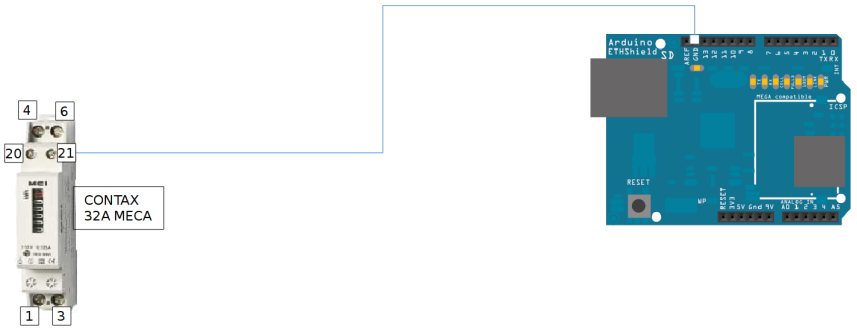

Figura 4.8: Collegamento dal foro 21 al GND di Arduino.

Si necessita poi di tagliare un jumper maschio-maschio di Arduino, saldare un'estremità sguainata del jumper con una della resistenza e, sulle stesse parti saldate, saldare l'altra estremità tagliata del jumper.

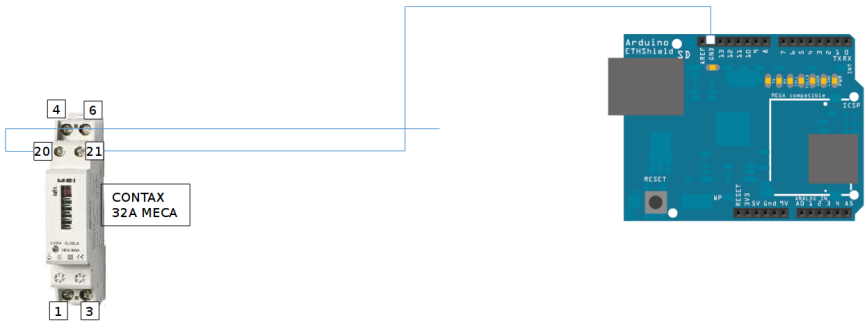

Figura 4.9: Tagliare e saldare come in schema.

In questo modo possiamo poi fissare un'estremità del jumper al 20 del contatore e l'altra estremità del jumper al pin 2 o 3 di Arduino.
Infine, si deve tagliare un altro jumper maschi-maschio Arduino,

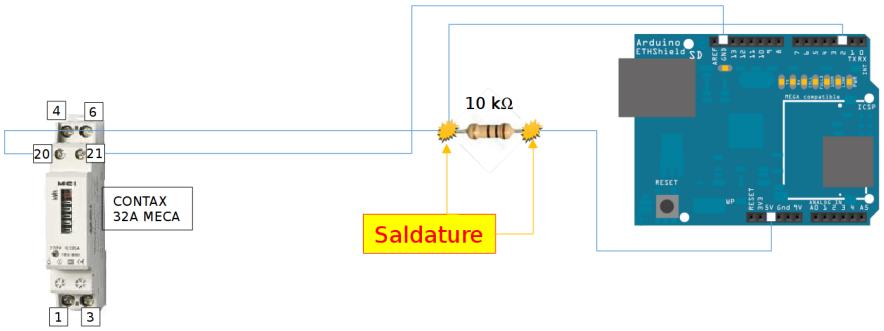

Figura 4.10: Schema finale.

saldare un'estremità con l'altra estremità libera della resistenza e inserire l'estremità del jumper appena saldato nel pin 5V della Board.

Connessione hardware tra Relay ed Arduino UNO

La connessione tra cavo e Relay Arduino è molto più semplice. Infatti basta tagliare il cavo di fase uscente dal contatore (quindi quello in uscita dal foro 3) e collegare la parte entrante nel Relay con il "Comune" del Relay (ovvero con l'ingresso in mezzo tra i tre disponibili per ogni blocco del Relay, che ricordiamo è composto da due blocchi Relay) e l'altra estremità tagliata con l'uscita "Normalmente Aperto".

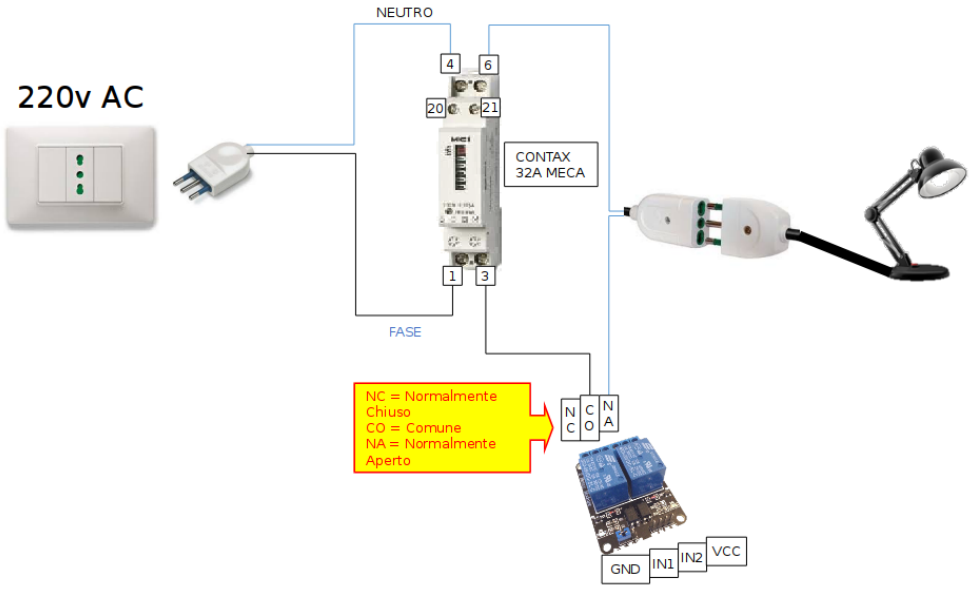

Figura 4.11: Schema collegamento.

Poi Basta collegare il GND del Realy con quello dell'Arduino, IN1 del Relay con il pin 2 di Arduino, IN2 con il pin 3 e VCC del Relay con i 5V di Arduino, utilizzando dei jumper maschio-femmina.