UNIVERSITÀ degli STUDI di CATANIA

DIPARTIMENTO di INGEGNERIA ELETTRICA, ELETTRONICA e INFORMATICA

CORSO DI LAUREA IN INGNEGNERIA INFORMATICA

LILLY CUTAIA

MONITORAGGIO REMOTO DI UN ROBOT AUTONOMO

Relatore:
Prof. Salvatore Monteleone

ANNO ACCADEMICO 2017-2018

A mia sorella Giusy,

spero che almeno se la legga

INDICE

Introduzione

L'obiettivo di questa tesi è quello di realizzare un robot autonomo in grado di operare anche in situazioni difficili e senza l'ausilio diretto dell'uomo. A causa della complessità del progetto, che avrebbe richiesto senz'altro anni di lavoro, in questo elaborato si è deciso di approfondire solo uno dei tanti aspetti che avrebbero potuto essere trattati, e cioè l'aspetto software che riguarda direttamente la gestione e l'elaborazione dei dati trasmessi dal robot.

È stato quindi necessario simulare le parti non presenti fisicamente, e cioè il robot in sé e la sua Intelligenza Artificiale: per sopperire a queste mancanze è stato utilizzato un microcontrollore Arduino, grazie al quale è stato possibile effettuare le successive prove senza intoppi.

Un sistema di questo tipo, infine, ha senz'altro innumerevoli applicazioni: potrebbe essere utilizzato infatti per monitorare continuamente zone impervie o disabitate, magari per vedere se sono idonee all'insediamento umano; potrebbe essere utile dopo una catastrofe naturale, per monitorare il territorio in sicurezza e autonomia; un robot simile potrebbe infine essere inviato nello spazio o sul fondo del mare, per acquisire dati utili sulla conformazione geografica e sul clima.

Le applicazioni sono innumerevoli anche perché, come sarà in seguito spiegato e approfondito, il codice è estremamente modulare: basta infatti aggiungere o modificare i campi appositi all'interno dei file designati per avere informazioni su tutto ciò che si vuole. In questo progetto è stato usato un sensore di temperatura per la sua facilità di utilizzo e di reperimento e per il basso costo,

ma nulla vieta di utilizzare qualsiasi altro tipo di sensore o di modulo disponibile in commercio.

Capitolo 1 – Storia della Robotica

Grazie all'avanzamento tecnologico degli ultimi anni l'uomo è riuscito a sviluppare diversi modelli di robot, utili nella vita di tutti i giorni. Inviati nello spazio alla scoperta di nuovi pianeti, fino all'esplorazione dei più profondi fondali marini, i robot ormai fanno parte della vita quotidiana di ognuno di noi, e il loro utilizzo trova innumerevoli applicazioni in ambito scientifico, industriale, medico e anche domestico.
Un importante aspetto che sarà di seguito trattato è quello dei robot autonomi, cioè in grado di svolgere compiti, più o meno elementari, senza l'ausilio dell'uomo. Ne sono un esempio le sonde inviate su Marte, in grado di raccogliere e analizzare autonomamente frammenti prelevati dal suolo, o i grandi macchinari industriali che sono in grado di autoregolarsi e creare il prodotto finito a partire dalla materia prima senza un controllo diretto da parte dell'uomo.

In generale, *"un robot è un'apparecchiatura artificiale che compie determinate azioni in base ai comandi che gli vengono dati e alle sue funzioni, sia in base ad una supervisione diretta dell'uomo, sia autonomamente basandosi su linee guida generali, magari usando processi di intelligenza artificiale."* [6]

Il robot quindi esula dal controllo umano diretto, venendo di fatto programmato attraverso un computer che può essere interno o esterno al robot stesso, o attraverso un microcontrollore. La parte hardware è invece composta da attuatori, motori, e sensori che gli permettono di percepire uno stimolo

proveniente dall'ambiente esterno, elaborarlo attraverso il programma, e agire di conseguenza. A tal proposito è doverosa la distinzione tra:

- Robot Autonomi;
- Robot Non Autonomi.

Robot Autonomi

Sono senz'altro la categoria più avanzata poiché operano attraverso algoritmi molto complicati che fanno uso di tecniche di intelligenza artificiale, come ad esempio reti neurali e apprendimento automatico. Un esempio sono i robot tagliaerba, in grado di sapere autonomamente quale punti tagliare e quando tornare alla base per ricaricarsi, oppure gli "animatronic" dell'industria cinematografica, atti a simulare in maniera più verosimile possibile i movimenti naturali degli animali a partire dai quali sono stati realizzati.

Robot non Autonomi

Sono guidati da un software deterministico che consente loro di effettuare operazioni ripetitive in modo autonomo con un controllo minimo da parte dell'uomo. Ne sono un esempio i robot utilizzati per lavorare in ambienti ostili o nelle industrie.

Questo utilizzo così massiccio di macchine autonome apre anche nuovi scenari di applicazione: esse possono infatti essere adoperate anche durante calamità e disastri naturali, per esempio per raccogliere in sicurezza dati su zone particolarmente inquinate o dove è presente materiale radioattivo.

Nei seguenti capitoli verranno approfonditi i dettagli della realizzazione dal punto di vista software e hardware, di un modello di robot in grado di raccogliere dati sull'ambiente circostante.

Capitolo 2 – L'Hardware

Il "cuore" del robot: il microcontrollore

La periferica principale che verrà utilizzata è Arduino, una board in grado di leggere degli input e di produrre un output a partire da essi. Una serie di istruzioni scritte nel linguaggio di programmazione che caratterizza Arduino stesso vengono, infatti, inviate al microcontrollore che le esegue [4].

La scheda prevede uno slot per Micro SD, un jack Ethernet ed un connettore di tipo USB-A Host. L'Arduino Yún è anche dotato di un modulo WiFi integrato per la connessione ad un router wireless, o per agire come access point. La scheda può essere alimentata solamente attraverso il connettore micro-USB o il pin Vin, inoltre, non include un regolatore di tensione a 5V: alimentando la scheda con più di 5V (ad es. attraverso il pin Vin) si rischia di danneggiarla. [5]

Considerando il peso e le dimensioni ridotte di Arduino e la possibilità di essere alimentato anche attraverso una batteria collegata direttamente al pin Vin, potrà essere posizionato sul modello di robot che si deciderà di utilizzare, sia esso una macchinina radiocomandata o un drone. Trovandosi direttamente a bordo infatti, potrà direttamente elaborare e inviare i dati al software, che si occuperà in seguito di catalogarli e salvarli, per renderli fruibili all'utente finale.

Modulo GPS, Giroscopio e Barometro

A prescindere dall'enorme quantità di modelli che si trovano in commercio, la funzione del modulo GPS è sempre quella di inviare la posizione di un oggetto attraverso coordinate quali latitudine e longitudine. In questo progetto, che di fatto non è stato realizzato a livello hardware, non useremo nessun sensore GPS in particolare, poiché le coordinate sono state simulate. Qualora si volesse procedere alla realizzazione fisica dello stesso, sarebbe comunque facile interfacciare la board con questi moduli poiché Arduino dispone già di librerie in grado di estrapolare le coordinate satellitari, convertirle e inviarle alla seriale, dalla quale possono essere direttamente utilizzate.

Se il progetto venisse realizzato attraverso l'ausilio di un drone, sarebbe utile inserire anche un giroscopio. Questo è un sensore non molto costoso in grado di percepire le variazioni della velocità angolare di un oggetto che si muove nello spazio. Potrebbe quindi rivelarsi particolarmente utile qualora si dovesse stabilizzare un UAV che vola in condizioni climatiche particolarmente avverse. A seconda delle variazioni di velocità angolare e posizione dello stesso, infatti, si potrebbe immediatamente procedere a correggerne il volo anche tramite l'ausilio del radiocomando. Solitamente i droni che si trovano in commercio hanno già un giroscopio integrato che rende tutte le operazioni sopra descritte

quasi automatiche, ma potrebbe essere utile avere un ulteriore giroscopio affinché se ne possa programmare il funzionamento a proprio piacimento.
Ultimo ma non meno importante, il barometro (o misuratore di pressione) in grado di dire a che altezza dal suolo sta volando il nostro drone. Pur non potendone trascurare l'enorme utilità in certi casi, bisogna dire che i barometri sono sensori un po' più costosi rispetto ai giroscopi, oltre al fatto che aggiungono peso al nostro sistema, per cui, prima di valutarne l'utilizzo, bisogna fare un'accurata analisi riguardo il peso trasportabile, durata della batteria, e quello che il drone effettivamente andrà a misurare. Sarà infatti fondamentale qualora volessimo ad esempio andare a valutare la variazione di temperatura a differenti altitudini, sarà perfettamente inutile invece nel caso si stia utilizzando un UGV, o robot terrestre.

Sensore di Temperatura

Dato che lo scopo del progetto è principalmente quello di misurare le variazioni di temperatura di una regione attraverso un robot, si è reso necessario l'acquisto e il conseguente utilizzo di un sensore di temperatura. Di questi sensori, dal costo di pochi euro, ne esistono un'infinità di marche e modelli diversi. Il loro funzionamento si basa sul fatto che la resistività di un metallo cresce linearmente al crescere della temperatura [7] secondo la formula:

$$R(T) = R_0 (1+\alpha T)$$

dove R(T) è la resistenza di uscita (funzione della temperatura), R_0 è il valore di resistenza misurato a zero gradi, T è la temperatura e α (alfa) è un coefficiente di proporzionalità che dipende dal sensore. Solitamente sono realizzati in metalli come il nichel, il platino o il rame, sono molto precisi e stabili. Quando

vengono attraversati da una corrente, la variazione di resistenza viene trasformata in una variazione di tensione che produce effettivamente la misura, con qualche piccolo errore dovuto all'effetto Joule.

Per questo progetto è stato utilizzato un sensore di temperatura di tipo LM35, facente parte della famiglia dei sensori di temperatura integrati. Questi sensori infatti possiedono al loro interno anche appositi circuiti atti ad amplificare il segnale e a linearizzarlo. Esso presenta inoltre un'uscita in tensione direttamente proporzionale alla temperatura misurata in gradi Celsius secondo la formula:

$$V = K \cdot T$$

dove V è la tensione prodotta, T è la temperatura misurata (espressa in gradi Celsius, °C) e K è una costante moltiplicativa di valore 10 mV/°C.

Lo schema di utilizzo è molto semplice e mostrato in figura:

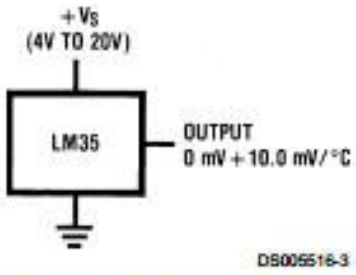


FIGURE 1. Basic Centigrade Temperature Sensor (+2˚C to +150˚C)

Questo modello di sensore è abbastanza economico ed estremamente efficace ed affidabile: durante una delle prove infatti è stato per errore montato al contrario, e, nonostante sia diventato incandescente, una volta fatto raffreddare e rimontato nel verso giusto, ha continuato a funzionare esattamente come prima, fornendo indicazioni corrette sulle misure rilevate.

Capitolo 3 – Il Software

Il software è senz'altro la parte più corposa di questo progetto, poiché gli algoritmi di gestione sono stati tutti correttamente implementati e testati. Il lavoro si è quindi concluso con la realizzazione di diversi programmi in grado di assolvere singolarmente ad uno specifico compito e che, integrati tutti insieme, hanno dato vita al progetto. Un possibile sviluppo e/o continuazione del lavoro potrebbe essere quello di unirli tutti insieme per creare un singolo programma, magari anche con una interfaccia utente, che consenta di caricare, scaricare e graficare adeguatamente i dati richiesti.
Tra linguaggi di programmazione utilizzati troviamo:

- C/C++ su IDE Arduino;
- GeoJSON;
- MySQL;
- PHP;
- HTML, CSS e Javascript.

Sono stati utilizzati strumenti e programmi totalmente freeware e gratuiti, tra i quali l'IDE Arduino [8], MySQLWorkbench [9], XAMPP [10] per implementare un server locale, Notepad++ [11] per scrivere la parte di interfaccia WEB. I dati sono inoltre stati caricati su un canale ThingSpeak per una possibile futura interfaccia con il programma di calcolo Matlab.

Arduino e il suo Linguaggio di Programmazione

L'IDE di Arduino è un'applicazione scritta in Java derivata dall'IDE creato per il linguaggio di programmazione Processing e per il progetto Wiring [12].
L'ambiente di sviluppo è abbastanza intuitivo, ed è possibile utilizzare un'infinità di librerie che rendono abbastanza agevole l'interfacciamento con moduli e/o sensori esterni e con altre piattaforme, tra le quali ad esempio ThingSpeak. Attraverso l'IDE si può compilare e in seguito caricare il codice sulla board semplicemente premendo un tasto, e vedere eventuali messaggi scritti sul monitor seriale.
La versione utilizzata in questo progetto è la 1.8.8, abbastanza stabile ma non del tutto esente da bug abbastanza fastidiosi: l'IDE è stata testata su macchine di ultima generazione che montano il sistema operativo Windows 10, e non di rado avvenivano freeze dell'IDE, che smetteva di riconoscere la scheda o di compilare e inviare un programma assolutamente corretto senza alcun apparente motivo, o a volte diceva di non trovare la scheda collegata alla porta a cui era effettivamente collegata. L'unico modo per risolvere era chiudere tutte le finestre e riavviare l'IDE, che così riprendeva a funzionare. Nulla di compromettente comunque, una volta imparato il trucco, al minimo segnale di instabilità si salvava tutto e si riavviava.
Per poter creare un eseguibile devono essere definite due funzioni di tipo `void`:

- `void setup()`: da utilizzare solo una volta all'inizio, serve a definire le variabili e le impostazioni iniziali che rimarranno invariati per tutta l'esecuzione del programma;

- `void loop()`: funzione che viene invece invocata ripetutamente, per cui è al suo interno che va messo il codice che deve essere eseguito ciclicamente. La sua esecuzione, infatti, è interrotta solo se si toglie l'alimentazione alla scheda.

Il codice Arduino realizzato per questo progetto è riportato in Appendice 1.

Oltre alle librerie necessarie per il funzionamento dell'applicazione e il collegamento alla piattaforma ThingSpeak, è stato utile realizzare una libreria personalizzata, denominata `secret.h`. Questa libreria è stata creata appositamente per rendere invisibili le credenziali del canale ThingSpeak, che è un canale privato. Anche l'indirizzo IP della macchina alla quale si stanno inviando i dati è stato volutamente cancellato. Queste infatti sono informazioni sensibili che si preferisce non rivelare, ma, ad ogni modo, sostituendo il proprio IP alla `get` e le credenziali del proprio canale ThingSpeak al posto dei valori `SECRET_CH_ID` e `SECRET_WRITE_APIKEY`, il programma funzionerà correttamente.

Le librerie `Bridge`, `BridgeClient` e `BridgeHTTPClient` sono state utilizzate per inviare i dati alla pagina `insert.php`. All'interno della funzione `void setup()`, oltre ad inizializzare il Bridge e la porta seriale alla frequenza di 9600 Baud, è stato anche scritto un codice in grado di prelevare l'indirizzo MAC dell'Arduino. Questo è molto utile poiché, nel caso ci fosse più di un robot, in questo modo non solo si possono identificare tutti in maniera sicura (il MAC Address è caratteristico della scheda di rete, non è possibile cambiarlo ed è univoco) ma, nel caso di avaria, misure errate o qualsivoglia imprevisto, si può facilmente individuare il robot "rotto" e procedere alle dovute riparazioni o

sostituzioni. Dato che il robot è autonomo, infine, ci permette anche di capire quale robot sta andando in quale direzione.

Data l'assenza di un modulo GPS, si è ritenuto necessario simulare un possibile percorso effettuato dal robot. A tal proposito sono state utilizzate le coordinate della linea bus 101 dell'azienda trasporti pubblici AMT della città di Catania [13]. È stato quindi creato un file ad hoc in formato `txt` dal quale fosse facilmente possibile estrarre le coordinate di latitudine e longitudine, salvarle all'interno di omonime variabili, ed associarle all'indirizzo MAC dell'Arduino e ad un determinato valore di temperatura, estratto tramite l'apposito sensore montato sul PIN A5 (analogico).
Una volta fissate tutte le variabili, all'interno della funzione `client2.get()` viene creata una stringa sempre diversa per ogni iterazione del ciclo *while* iniziale, che si occupa di inviare queste informazioni alla pagina `insert.php`, che, se tutto è stato ricevuto correttamente, a sua volta andrà a salvare le informazioni su un database di tipo MySQL.
Infine, le stesse informazioni vengono inviate ad un canale ThingSpeak che produce un messaggio di errore sulla seriale nel caso qualcosa fosse andato storto, un messaggio di conferma sempre sulla seriale nel caso invece fosse andato tutto a buon fine. Poiché ThingSpeak può accettare dati solo ogni 15 secondi, il codice esegue ogni 15 secondi: ciò implica che ogni 15 secondi viene effettuata una misurazione con conseguente invio dei dati al PHP e al canale ThingSpeak.

Questo codice è strutturato in maniera tale da funzionare attraverso una connessione WiFi: Arduino Yún infatti possiede un'interfaccia WiFi integrata che può essere configurata con qualsiasi connessione, quindi potenzialmente

anche con un hotspot mobile. Con le opportune modifiche si potrebbero usare anche un modulo Bluetooth o Zigbee, ma questo andrebbe ad aumentare senz'altro i costi e il peso del robot.

Cosa è GeoJSON?

GeoJSON [1] viene utilizzato per codificare strutture dati geografiche sfruttando il formato JSON. Queste strutture dati sono pensate per rappresentare elementi (punti, linee, poligoni, etc.) all'interno di una mappa, quindi, attraverso questo formato possiamo leggere e indicare su una mappa punti di interesse e i dati eventualmente ad essi associati come, ad esempio, la temperatura misurata in un certo istante in un determinato punto.
Un esempio di codice GeoJSON è riportato qui di seguito. Esso rappresenta la struttura che si andrà ad utilizzare per i file di questo progetto.

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {
        "Temp (°C)": 25.8
      },
      "geometry": {
        "type": "Point",
        "coordinates": [
          15.073649883270262,
          37.526277219935196
```

```
        ]}
},
 {
      "type": "Feature",
      "properties": {
        "Temp (°C)": 25.9
      },
      "geometry": {
        "type": "Point",
        "coordinates": [
          15.074020028114319,
          37.52639634250274
        ]} },
    {
      "type": "Feature",
      "properties": {
        "Temp (°C)": 25.9
      },
      "geometry": {
        "type": "Point",
        "coordinates": [
          15.074374079704283,
          37.52651546488007
        ] }
 },
    {
      "type": "Feature",
```

```
      "properties": {
        "Temp (°C)": 26.0
      },
      "geometry": {
        "type": "Point",
        "coordinates": [
          15.07471740245819,
          37.52663884142749
      ]}
}]}
```

Un oggetto GeoJSON può definire diversi elementi, tra i quali:

- Oggetti geometrici nello spazio;
- *Feature*;
- *FeatureCollection*, cioè raccolte di *Feature*.

Oggetti Geometrici nello Spazio

Possono essere punti, linee o superfici, e nel GeoJSON RFC vengono definiti ben sette tipi di figure geometriche, che sono: *Point*, *MultiPoint*, *LineString*, *MultiLineString*, *Polygon*, *MultiPolygon*, e *GeometryCollection*.

Prima di definire in dettaglio cosa sono queste figure, cosa rappresentano e a cosa servono, deve essere compreso il concetto di *Position* (posizione), intesa come un array di due o tre numeri: i primi due rappresentano latitudine e longitudine, il terzo rappresenta l'altitudine, ma è un parametro opzionale. In questo caso verrà inserito soltanto se il robot sarà in grado di volare.

Nella tabella sottostante sono elencati tutti i formati per le diverse API, e le API stesse.

lon, lat	lat, lon
formats	
• GeoJSON • KML • Shapefile • WKT • WKB • geobuf	• GeoRSS • Encoded Polylines (Google)
javascript apis	
• OpenLayers • d3 • ArcGIS API for JavaScript • Mapbox GL JS	• Leaflet • Google Maps API
mobile apis	
• Tangram ES[1]	• Google Maps iOS/Android • Apple MapKit
service specifications	
• WFS 1.0.0 [1] • WMS 1.1.1 [1]	• WFS 1.1.0 & 2.0.0 [1] • WMS 1.3.0 [1]
misc	
• OSRM[1] • Redis[1]	

I *Points* e i *Multipoints* differiscono tra loro per il fatto che, mentre i *Points* sono rappresentati da una sola coppia di coordinate, per i *Multipoints* troviamo un array di oggetti di tipo *Position*, cioè un array di posizioni. Da ciò si deduce il fatto che è anche possibile creare dei segmenti indicando solamente il punto di inizio e di fine della linea. Analogamente, possiamo creare più linee che si dipartono dallo stesso punto e *Polygons* (poligoni), che non sono altro che multilinee chiuse a formare appunto un poligono.

Features e FeatureCollections

Una *Feature* è un oggetto che viene usato come contenitore, infatti, come si può vedere dal codice riportato in figura, una volta definita una *Feature* andranno scritti al suo interno tutti i parametri che la caratterizzano.

```
{
      "type": "Feature",
            "properties": {
                  "name": "Cittadella Universitaria"
            },
      "geometry": {
            "type": "Point",
            "coordinates": [
                  15.072920322418215,
                  37.5262687111731
            ]
      }
}
```

È possibile anche avere *FeatureCollection*, che come dice la parola stessa sono collezioni di *Feature*: consistono in un package contenente diverse *Feature*, ognuna con la sua definizione e i suoi parametri.

Il codice PHP deputato a creare il nostro file GeoJSON, andrà ad inglobare tutti i dati proprio in una *FeatureCollection*.

Brevi cenni su JSON

JSON (JavaScript Object Notation) [3] è il formato utilizzato nella specifica GeoJSON per descrivere i dati che essa tratta. JSON è un formato molto utilizzato per lo scambio di dati e informazioni: quando un client e un server devono scambiare dati infatti, gli oggetti JavaScript vengono serializzati in formato JSON che viene inviato al server tramite particolari chiamate denominate Ajax. La deserializzazione dal formato JSON permette di riconvertire i dati ricevuti in oggetti Javascript che possono quindi essere elaborati dal sistema. La sintassi è molto semplice, con dati costruiti come coppie di proprietà/valori separati da una virgola. JSON ammette solo valori semplici e atomici tra i quali stringhe, numeri, array, oggetti letterali, null o valori booleani (true o false) [2].

Brevi cenni su MySQL

MySQL è un RDBMS (Relational Database Management System) composto da un client e un server. Affinché vi si possano salvare i dati all'interno infatti è necessario interfacciare il programma MySQLWorkbench con un server, sia esso locale (emulato attraverso il programma XAMPP) oppure remoto. I dati verranno quindi salvati in tabelle conservate su questi server che possono essere scaricate in qualsiasi momento accedendovi. MySQL è uno strumento estremamente versatile poiché è in grado di interfacciarsi con innumerevoli linguaggi di programmazione quali ad esempio il già citato PHP, Java, Python e così via, il che lo rende estremamente appetibile per la gestione di database che fanno riferimento a siti WEB dinamici.

La sintassi inoltre è estremamente semplice e diretta, più vicina alla lingua inglese parlata che ad un linguaggio di programmazione vero e proprio, e

consente di memorizzare trigger e procedure per una gestione più diretta e dinamica delle relazioni.

In questo progetto ne è stato fatto un uso estremamente semplice rispetto alle sue enormi potenzialità, ma funzionale al compito richiesto: creare un database con all'interno una tabella da riempire con i valori inviati da Arduino al PHP. Di seguito il codice per la creazione del DB e della tabella:

```
1. CREATE DATABASE test_arduino;
2. USE test_arduino;
3. CREATE TABLE DATI (
4.   lat     VARCHAR(50) NOT NULL,
5.   lon     VARCHAR(50) NOT NULL,
6.   temp    INT NOT NULL,
7.   mac     VARCHAR(50) NOT NULL)
8. ENGINE = 'InnoDB';
```

Come si evince dall'immagine, all'interno della tabella DATI è stato inserito lo spazio per Latitudine (`lat`), Longitudine (`lon`), Temperatura (`temp`) e Indirizzo MAC dell'Arduino (`mac`).

La tabella è stata in seguito riempita con diverse *entry*, come mostrato in figura. In questo caso si disponeva di un solo Arduino, per cui l'indirizzo MAC sarà sempre lo stesso. Se ci fossero stati più microcontrollori invece, si sarebbe visto un MAC Address diverso per ogni microcontrollore.

La query `select *` consente di stampare tutte le *entry* presenti all'interno di una determinata tabella contenuta in un database.

Il nome della tabella in questo caso è denominata `dati` e si trova nel database chiamato `test_arduino`.

La query effettuata dal file `create_file.php` per creare il file GeoJSON, sarà effettuata proprio utilizzando queste *entry*.

```
1 •   select * from test_arduino.dati;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

lat	lon	temp	mac
37.531746263079	15.108542391367	294	B4:21:8A:F0:6F:31
37.531692490655	15.108448426337	290	B4:21:8A:F0:6F:31
37.531672283811	15.10833309135	288	B4:21:8A:F0:6F:31
37.531737158398	15.107561956258	285	B4:21:8A:F0:6F:31
37.531872224979	15.106132338853	283	B4:21:8A:F0:6F:31
37.532046174333	15.10428572566	280	B4:21:8A:F0:6F:31
37.532178049822	15.102923163481	279	B4:21:8A:F0:6F:31
37.532182303866	15.102445730276	277	B4:21:8A:F0:6F:31
37.53216741471	15.102142640657	277	B4:21:8A:F0:6F:31
37.532129128297	15.101871737547	275	B4:21:8A:F0:6F:31
37.532044047307	15.101471330137	275	B4:21:8A:F0:6F:31
37.531929187817	15.101083168097	273	B4:21:8A:F0:6F:31
37.531776041556	15.100667425699	274	B4:21:8A:F0:6F:31
37.531695214235	15.100460895605	272	B4:21:8A:F0:6F:31
37.531569719012	15.100187310286	269	B4:21:8A:F0:6F:31
37.531284695177	15.099782296724	268	B4:21:8A:F0:6F:31
37.530997543197	15.099420198507	267	B4:21:8A:F0:6F:31
37.530323263462	15.098738917418	267	B4:21:8A:F0:6F:31
37.529999947357	15.098435827799	268	B4:21:8A:F0:6F:31
37.529727680077	15.098095187254	265	B4:21:8A:F0:6F:31
37.529440522103	15.097679444857	267	B4:21:8A:F0:6F:31
37.529236320204	15.097325393267	266	B4:21:8A:F0:6F:31
37.529117202172	15.097065218992	266	B4:21:8A:F0:6F:31
37.528983194158	15.096705802984	267	B4:21:8A:F0:6F:31

dati 1 ×

Brevi cenni su PHP

PHP (acronimo ricorsivo di "PHP: Hypertext Preprocessor", preprocessore di ipertesti; originariamente acronimo di “Personal Home Page” è un linguaggio

di scripting interpretato, originariamente concepito per la programmazione di pagine web dinamiche [14].

È molto versatile, poiché consente anche un paradigma di programmazione Object Oriented, ed è completamente compatibile con innumerevoli DMBS, tra i quali il già citato MySQL (utilizzato in questo progetto), MariaDB, Oracle, MongoDB e molti altri.

Una funzionalità particolarmente interessante è quella di poter includere dei file all'interno di altri file: in questo modo sicuramente si mantengono pulizia e leggerezza del codice, dato che non c'è la necessità di ridefinire le variabili o riscrivere porzioni di programma.

In questo progetto si è fatto uso di tre file PHP ognuno per assolvere ad una determinata funzione:

db.php

```php
<?php

//qui seleziono il mio database per potermi connettere ad esso , infatti questi parametri vengono passati ogni volta che
//viene effettuata una connessione, alla funzione mysqli_connect() che instaura la connessione vera e propria con il db

$servername = "localhost";
$user = "root";
$pass = "";
$db_name= "test_arduino";

?>
```

In questo file sono stati passati i dati per la connessione al nostro DB. Dato che si stava utilizzando un server locale emulato attraverso XAMPP, è bastato definire il server come "*localhost*" e passare alle altre variabili di autenticazione i dati di MySQLWorkbench. Lo scopo di questo codice quindi è soltanto quello di instaurare una connessione tra il codice PHP e il database MySQL. È stato

creato un file apposito per rendere più semplici le eventuali modifiche delle credenziali del database o del server utilizzato, dato che, potenzialmente, gli script potrebbero essere trasferiti su un server remoto sempre connesso.

insert.php

```
1. <?php
2. session_start();
3. include("db.php");
4. $conn= mysqli_connect($servername, $user, $pass, $db_name);
5.
6. if (isset($_GET["lat"]) && isset($_GET["lon"]) && isset($_GET[
   "temp"]) && isset($_GET["mac"])) {
7.
8.     $lat  = $_GET["lat"];
9.     $lon  = $_GET["lon"];
10.    $temp = $_GET["temp"];
11.    $mac = $_GET["mac"];
12.
13.    mysqli_query($conn, "INSERT INTO dati(lat, lon, temp, mac)
    values ('$lat', '$lon', '$temp', '$mac')");
14.
15.
16.}
17.?>
```

In questo codice si può vedere per la prima volta come avviene la connessione e la comunicazione tra il file PHP e il DB MySQL. Attraverso la funzione `session_start()`, infatti, viene avviata una sessione, incluso il file `db.php` così da avere anche qui definite tutte le credenziali di DB e server, infine viene avviata la connessione vera e propria. Da questo momento in poi, il file PHP è come se fosse in ascolto: una volta connesso con il DB infatti, rimane in attesa di ricevere dati dall'esterno attraverso il metodo GET. In particolare, PHP permette la ricezione di dati attraverso il metodo GET e il metodo POST. La differenza principale tra i due metodi è che il GET richiede i dati tramite una *query string*, cioè una stringa contenente tutti i parametri da passare, mentre invece il POST invia i dati direttamente senza l'ausilio della *query string*, per questo motivo viene preferito ad esempio per l'invio di dati provenienti dalla

compilazione dei *form* [15]. In questo caso si è preferito utilizzare il metodo GET più che altro per la sua semplicità: è bastato infatti creare la *query string* dinamica sul codice Arduino e utilizzare il metodo `get` messo a disposizione dalla libreria `HttpClient` per far comunicare con successo le due parti del codice.

L'inserimento nel database è condizionato dall'effettiva ricezione di tutti i dati: la funzione `isset()` controlla che il GET abbia ricevuto dati validi e, se l'esito è positivo, allora la funzione `mysqli_query()` li inserisce tutti nel database.

create_file.php

Per semplicità, il codice verrà suddiviso in due parti, anche se fa parte dello stesso file. Prima verrà descritta la parte statica, in seguito ci si occuperà della parte dinamica di creazione delle *entry* da stampare sul file stesso.

```
1. <?php
2. session_start();
3. include("db.php");
4. $conn= mysqli_connect($servername, $user, $pass, $db_name);
5.
6. $result= mysqli_query($conn, "SELECT * FROM dati");
7.
8. $num_rows = mysqli_num_rows($result);
9.
10. $i=1;
11.
12. $map_file= 'map.js';
13. $handle = fopen($map_file, 'w') or die
14. ('Error, cannot open the file!');
15.
16. $intestazione = '{"type": "FeatureCollection",
17.                     "features": [';
18.
19. fwrite($handle, $intestazione);
```

La prima parte del codice include, come spiegato anche in precedenza, il file "db.php" atto ad effettuare la connessione tra questo file e il database. Viene fatta la *query* al database affinché possa ritornarci tutti i dati in esso presenti, e

viene salvato il numero di righe ritornate nella variabile `$num_rows`. Questo dettaglio è importante perché ha permesso di scrivere l'algoritmo di gestione dei terminatori per i blocchi di file generati da ogni *entry*. Per lo stesso motivo viene definita la variabile `$i`, successivamente viene creato il file che si chiamerà `map.js`. Ogni volta che sarà richiamato questo codice, verrà creato un nuovo file a partire da tutte le *entry* del database.

Anche il file `map.js` è costituito da due parti: l'intestazione, fissa, scritta sul file successivamente alla sua creazione, e il corpo, descritto dalla parte di codice successiva:

```
1. while($row = mysqli_fetch_assoc($result)){
2.
3.     $data= '
4.         {
5.         "type": "Feature",
6.         "properties": {
7.             "temperature": '.$row["temp"].',
8.             "mac_address": "'.$row["mac"].'"
9.         },
10.        "geometry": {
11.            "type": "Point",
12.            "coordinates": [
13.                '.$row["lon"].',
14.                '.$row["lat"].'
15.            ]
16.        }
17.        }';
18.
19.    fwrite($handle, $data);
20.
21.    if($i< $num_rows){
22.        $comma = ',';
23.        fwrite($handle, $comma);
24.    }
25.
26.    else {
27.        $term= ']}';
28.        fwrite($handle, $term);
29.
30.    }
31.
32.    $i= $i+1;
33.
34.
35.}
36.
```

```
37. fclose($handle);
38. mysqli_close($conn);
39.
40. ?>
```

È qui che effettivamente viene creato il file: tramite un ciclo *while* che manda in *loop* questa parte di codice finché non ci sono più *entry* nel database, vengono estratte le singole righe e i dati memorizzati nelle rispettive variabili.

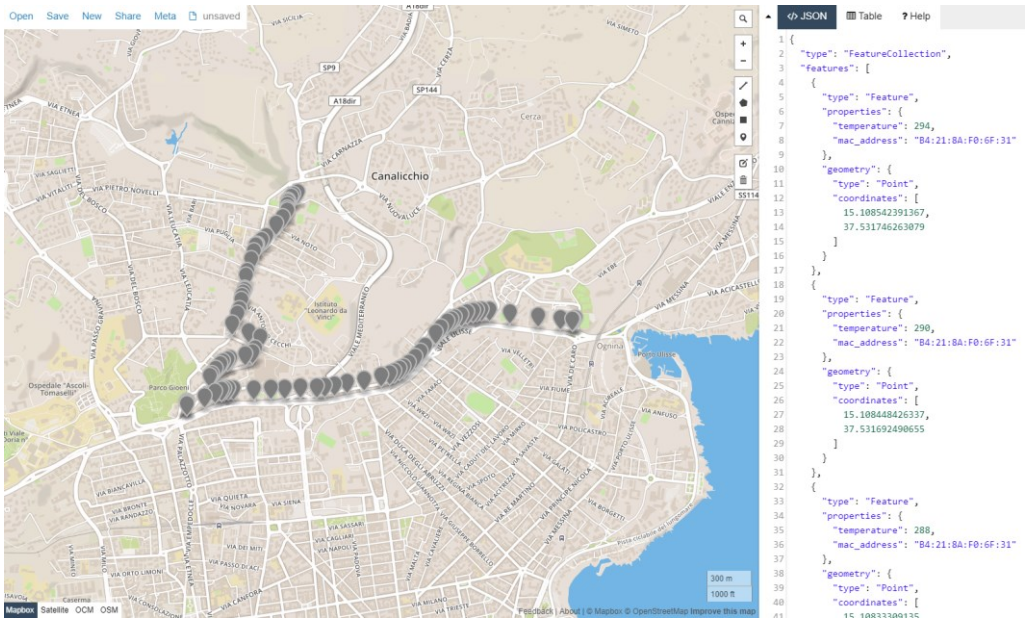
Per ogni *loop* ovviamente la riga elaborata e salvata sarà diversa, e, a seconda del numero di righe elaborate, viene inserito un diverso terminatore. GeoJSON infatti richiede che ogni *Feature* sia separata dalle altre attraverso una virgola, che viene quindi scritta sul file nel caso la riga elaborata in quel momento non sia l'ultima del database, quindi non ci siano più dati; se la riga elaborata è l'ultima, viene invece scritto il carattere terminatore del file, che chiude tutte le *Feature* e la *FeatureCollection*.

Questa gestione dei terminatori del file è stata effettuata attraverso il verificarsi di una condizione: se la variabile `$i`, inizializzata a `1`, è ancora minore del numero di righe restituite dalla funzione che conta le *entry* nel database, allora vuol dire che la *Feature* che si sta scrivendo in quel momento non è l'ultima, per cui viene inserito come terminatore una "virgola" (`$comma`), che prepara il file alla scrittura di un'altra *entry*; in caso contrario, se la variabile `$i` ha assunto un valore uguale al numero di righe, allora vuol dire che *l'entry* che si sta scrivendo in quell'istante è l'ultima, viene inserito il terminatore `$term` e il ciclo *while* si interrompe, producendo il file pronto all'uso. Ovviamente affinché questa seconda condizione avvenga, la variabile `$i` viene incrementata di `1` ad ogni ciclo, cosicché dopo un numero di cicli pari al numero di *entry* presenti sul database, il codice si interrompa e produca il file desiderato. Questo file, come si può facilmente vedere dal codice, ha struttura GeoJSON. Se lo si andasse a

caricare sul sito ufficiale GeoJSON.io, l'output sarebbe quello mostrato nella successiva figura, in cui:

Ad ogni "puntino grigio" segnato sulla mappa corrispondono:

- Coordinate (latitudine e longitudine);
- Indirizzo MAC dell'Arduino che ha inviato il dato;
- Temperatura misurata.



Ovviamente sarebbe scomodo caricare ogni volta il file creato sul sito GeoJSON.io, per questo motivo nel successivo capitolo si parlerà di come sia stata creata una pagina in linguaggio HTML in grado di ricevere autonomamente il file e disegnarlo sulla mappa.

Capitolo 4 – Interfaccia Grafica Web-based

Quando si parla di pagine WEB, HTML è la prima cosa che viene in mente. HTML è l'acronimo di HyperText Markup Language ("Linguaggio di contrassegno per gli Ipertesti") e non è un linguaggio di programmazione vero e proprio. Si tratta invece di un linguaggio di markup (di 'contrassegno' o 'di marcatura'), che permette di indicare come disporre gli elementi all'interno di una pagina [16]. Al suo interno possono trovarsi file .css e file .js, estensioni rispettivamente di CSS e di Javascript: il primo è un linguaggio utilizzato solo ed esclusivamente per definire la formattazione della pagina HTML in cui è inserito, strutturandone la grafica; il secondo invece è un semplice linguaggio di scripting orientato agli oggetti e agli eventi, poiché è in grado di eseguire codici diversi all'occorrenza di eventi diversi, quali ad esempio un click su un bottone o il passaggio del puntatore del mouse in una determinata zona della pagina. Tutti e tre questi strumenti vengono utilizzati insieme per gestire grafica, formattazione e comportamento delle più moderne pagine WEB, che hanno la caratteristica di essere altamente responsive e dinamiche.

In questo caso si vuole fornire all'utente finale una interfaccia user- friendly che gli consenta di analizzare i dati ricevuti nella maniera più semplice possibile. Per questo motivo è stata implementata una pagina chiamata "index.html", che si occupa di ricevere il file mappa e gestirlo adeguatamente, senza la necessità di caricarlo sul sito GeoJSON.io.

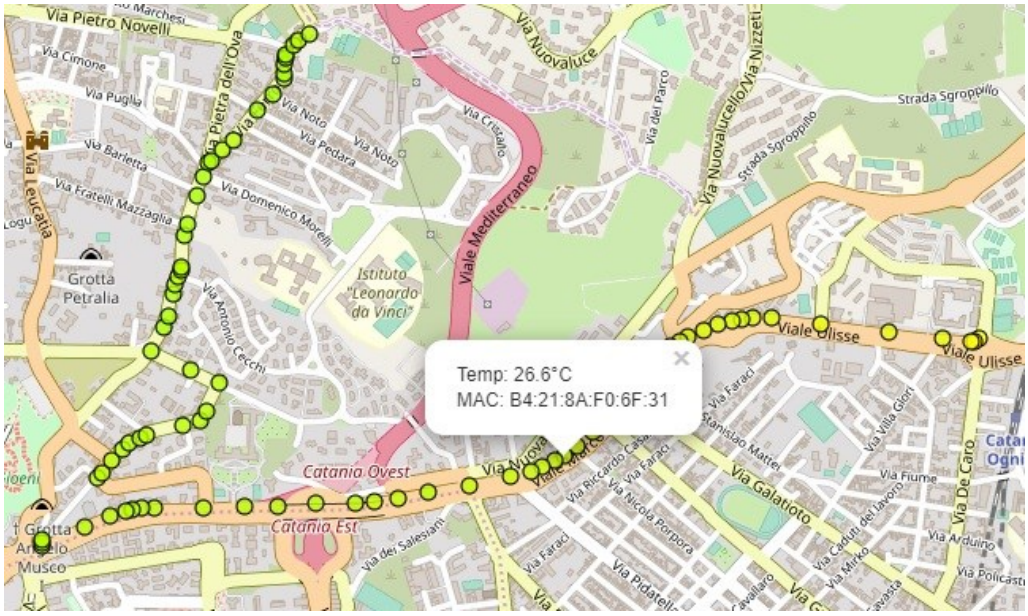
index.html

```
1. <!doctype html>
2. <html>
3.     <head>
4.         <link rel="stylesheet" href="./css/style.css" />
5.         <link rel="stylesheet" href="https://unpkg.com/leaflet
   @1.4.0/dist/leaflet.css" />
6.         <script src="https://unpkg.com/leaflet@1.4.0/dist/leaf
   let.js"></script>
7.     </head>
8.     <body>
9.
10.        <div id="map"></div>
11.
12.        <script src="./geojson/map.js"></script>
13.        <script src="./js/main.js"></script>
14.    </body>
15. </html>
```

Questa pagina è coadiuvata dal file `map.js` di cui si era già ampiamente discusso in precedenza, e dallo script `main.js`, di cui è riportato il codice in Appendice 2.

Dato che lo scopo era quello di creare una mappa "interattiva", si è reso necessario l'utilizzo della libreria `Leaflet`, una libreria open-source di Javascript in grado di creare mappe aventi titoli e marcatori personalizzati [17]. Gran parte del lavoro viene quindi svolto dal file `main.js`, che contiene un algoritmo in grado di associare ad ogni latitudine e longitudine un contrassegno avente come etichetta l'indirizzo MAC dell'Arduino e la temperatura misurata. È stato in seguito definito un *range* di colori associati ad ogni contrassegno, e che cambiano a seconda del valore di temperatura misurato: partendo dal colore verde (temperatura più bassa), indichiamo con colori sempre più accesi valori di temperatura più alti, fino al colore rosso. Vediamo che per misurazioni intermedie, anche il colore del contrassegno varia in maniera graduale.

Ciò che vede l'utente finale è una mappa di questo tipo:



Come si può notare dall'immagine, cliccando sui diversi contrassegni si possono vedere i dati ad essi associati, in accordo al file GeoJSON precedentemente creato. In questo caso le temperature sono tali che il *range* di colori dei contrassegni vada dal verde chiaro al giallo, con una temperatura media di 26°C. Creando ulteriori file GeoJSON attraverso lo script PHP precedentemente discusso e caricandoli nell'apposita cartella, la visualizzazione cambierà e si potrà vedere il nuovo percorso effettuato dal robot. Lo script è stato programmato per funzionare con file GeoJSON composti da un oggetto *FeatureCollection* avente all'interno una serie di *Feature*. Se invece si è interessati a lavorare solo sul percorso del robot, magari per vedere quale strada decide di fare un algoritmo di intelligenza artificiale in determinate condizioni, si potrebbe modificare leggermente lo script per far tracciare solo una linea sulla mappa.

Ad ogni modo, ogni modifica sostanziale avrà implicazioni ed applicazioni diverse per il progetto, e sarà funzionale al compito da assolvere.

Capitolo 5 – Interfacciamento con Thingspeak

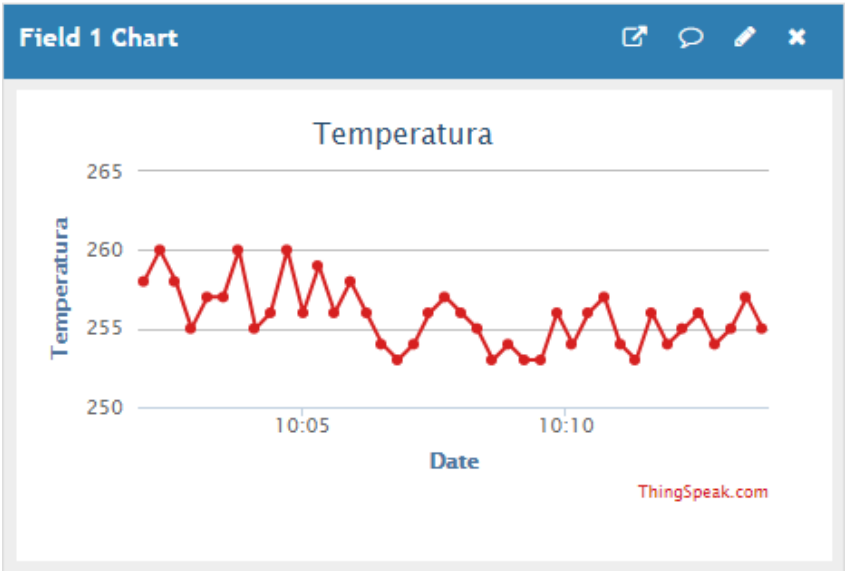
ThingSpeak è un servizio online che permette di visualizzare e analizzare dati salvati sul *cloud*. Una delle cose più interessanti è che i dati possono essere inviati direttamente dal *device* al canale di riferimento, senza bisogno di tramiti, e che i dati estratti possono essere interfacciati ed elaborati su Matlab, attraverso il quale è possibile anche effettuare analisi di carattere statistico e predittivo. Proprio grazie alla sua immediatezza, ThingSpeak è oggi utilizzato per l'analisi dei dati di innumerevoli applicazioni IoT, e per allenare algoritmi di intelligenza artificiale fornendo un set di dati completo in ogni sua parte.
In questo progetto non è stata implementata anche la parte relativa allo studio dei dati in Matlab, ma ci si è fermati a caricare i dati su un canale privato ThingSpeak dal nome "Misurazioni". All'interno di questo canale possiamo trovare quattro *fields*, relativi ognuno ad uno specifico dato inviato dall'Arduino. I quattro *field* quindi sono:

- **Temperatura**: salva i dati di temperatura;
- **Latitudine**: salva i dati relativi alla latitudine;
- **Longitudine**: salva i dati relativi alla longitudine;
- **Indirizzo MAC Arduino**: salva l'indirizzo MAC del microcontrollore.

A partire dai set di dati, ThingSpeak può anche creare dei grafici per mostrarci come varia quel determinato parametro nel tempo. Nella figura mostrata a pagina seguente troviamo il grafico di variazione della temperatura in un *range* di tempo lungo dieci minuti.

Dalla visualizzazione di ThingSpeak è possibile inoltre modificare questo grafico affinché mostri più o meno dati, e si possono inserire diversi tipi di personalizzazioni. Si può infatti cambiare la visualizzazione del grafico, modificarne i colori o esportarlo.

ThingSpeak, infine, consente anche di esportare i dati di un canale in formato JSON, così da poter essere elaborati e mostrati anche su pagine web. Le API infatti sono facilmente integrabili in un codice HTML.



Per quanto riguarda l'implementazione dal punto di vista di Arduino, è stato visto che questa è estremamente semplice poiché viene utilizzata la libreria `Thingspeak.h`, contenente già tutte le funzioni da richiamare per interfacciare il microcontrollore con il canale e inviarvi un set di dati.

Conclusioni

Nonostante questo elaborato contenesse le idee necessarie alla progettazione e alla successiva costruzione di un robot autonomo in grado di svolgere le funzioni presentate nell'introduzione, per brevità si è deciso di trattare in maniera approfondita solo uno degli innumerevoli aspetti che caratterizzano un progetto di questa portata, e cioè l'acquisizione e l'elaborazione dei dati.

Per questo motivo è stata posta particolare attenzione all'aspetto software del progetto più che all'aspetto hardware, soprattutto per la facilità di gestione e implementazione rispetto all'effettiva costruzione di un robot dotato di intelligenza artificiale propria. Inoltre, gli argomenti trattati focalizzano uno degli obiettivi principali di questo elaborato, e cioè la *modularità* del progetto: è stato realizzato un codice in grado di funzionare su qualsiasi dispositivo in grado di trasportare un Arduino, sia esso un robot autonomo o un semplice veicolo radiocomandato.

È stato necessario allora simulare un robot in grado di muoversi nello spazio acquisendo dati, i quali sono stati prima inviati ad un database e successivamente gestiti e mostrati all'utente finale attraverso pagine scritte rispettivamente in PHP e HTML.

Infine, è stata aperta una piccola parentesi sulla possibilità di gestione del database attraverso Thingspeak e il successivo interfacciamento con il software MATLAB, ma questo aspetto non è stato trattato in maniera approfondita poiché potrebbe essere ripreso e discusso in sviluppi futuri legati a questo progetto.

Appendice 1 – Codice Arduino

```
#include <ThingSpeak.h>
#include <Bridge.h>
#include "ThingSpeak.h"
#include "secrets.h"
#include <BridgeClient.h>
#include <HttpClient.h>
#include <FileIO.h>
#include <Process.h>

File linea;
BridgeClient client;
unsigned long myChannelNumber = SECRET_CH_ID;
const char * myWriteAPIKey = SECRET_WRITE_APIKEY;
String lat;
String longit;
int temp;
HttpClient client2;
String mac;

void setup() {

  Bridge.begin();
  Serial.begin(9600);
  ThingSpeak.begin(client);
  Process p;
  p.runShellCommand("ifconfig wlan0 | grep HWaddr | cut -d\" \" -f10");
  while (p.running());
  mac = p.readStringUntil('\n');


}

void loop() {
  File linea = FileSystem.open("/root/101.txt", FILE_READ);

  while (1) {
    lat = linea.readStringUntil(',');
    longit = linea.readStringUntil('n');

    temp = analogRead(A5);

    client2.get("insert_your_ip/insert.php?lat=" + lat + "&lon=" + longit + "&temp=" + temp + "&mac=" + mac);

    ThingSpeak.setField(1, temp);
    ThingSpeak.setField(2, longit);
    ThingSpeak.setField(3, lat);
    ThingSpeak.setField(4, mac);

```

```
    int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPI
  Key);
    if (x == 200) {
      Serial.println("Channel update successful.");
      delay(15000);
    }

    else
      Serial.println("There was an error updating channel");
  }
}
```

Appendice 2 – main.js

```
1. var map = L.map('map').setView([37.53253754448352, 15.09514175
   3554346], 15);
2.
3. L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.pn
   g', {
4.     attribution: '© <a href="https://www.openstreetmap.org/cop
   yright">OpenStreetMap</a> contributors'
5. }).addTo(map);
6.
7.
8.
9. L.geoJSON(data, {
10.     onEachFeature: function (feature, layer) {
11.         if (feature.properties && feature.properties.temperatu
   re && feature.properties.mac_address) {
12.             layer.bindPopup("Temp: " + feature.properties.temp
   erature/10
13.             + "°C<br>MAC: " + feature.properties.mac_address);

14.         }
15.     },
16.     pointToLayer: function (feature, latlng) {
17.         var green_to_red = [
18.             "#00FF00", "#11FF00", "#22FF00", "#33FF00",
19.             "#44FF00", "#55FF00", "#66FF00", "#77FF00",
20.             "#88FF00", "#99FF00", "#AAFF00", "#BBFF00",
21.             "#CCFF00", "#DDFF00", "#EEFF00", "#FFFF00",
22.             "#FFEE00", "#FFDD00", "#FFCC00", "#FFBB00",
23.             "#FFAA00", "#FF9900", "#FF8800", "#FF7700",
24.             "#FF6600", "#FF5500", "#FF4400", "#FF3300",
25.             "#FF2200", "#FF1100", "#FF0000"];
26.
27.         var geojsonMarkerOptions = {
28.             radius: 5,
29.             fillColor: green_to_red[Math.round(feature.propert
   ies.temperature/10*1.2-20)] || "#FF0000",
30.             color: "#000",
31.             weight: 1,
32.             opacity: 1,
33.             fillOpacity: 0.8
34.         };
35.
36.         return new L.circleMarker(latlng, geojsonMarkerOptions
   );
37.     }
38. }).addTo(map);
```

Bibliografia

[1] https://medium.com/@sumit.arora/what-is-geojson-geojson-basics-visualize-geojson-open-geojson-using-qgis-open-geojson-3432039e336d

[2] https://www.html.it/articoli/introduzione-a-json/

[3] https://www.w3schools.com/js/js_json_intro.asp

[4] https://www.arduino.cc/en/Guide/Introduction#

[5] Dispense del corso "Laboratorio di Architetture di Sistemi Fissi e Mobili" A.A. 2017/2018, C.d.S. in Ing. Informatica (L-8) – Dipartimento di Ingegneria Elettrica Elettronica e Informatica – Università degli Studi di Catania.

[6] https://it.wikipedia.org/wiki/Robot

[7] http://www.elemania.altervista.org/sensori/trasduttori/trasd4.html

[8] https://www.arduino.cc/en/main/software

[9] https://dev.mysql.com/downloads/workbench/

[10] https://www.apachefriends.org/it/index.html

[11] https://notepad-plus-plus.org/download/v7.6.5.html

[12] https://it.wikipedia.org/wiki/Arduino_(software)

[13] http://www.amt.ct.it/OpenData/linee.php?linea=101

[14] https://it.wikipedia.org/wiki/PHP

[15] https://www.html.it/pag/62463/le-richieste-http-get-e-post/

[16] https://www.html.it/pag/16026/introduzione22/

[17] https://leafletjs.com/

Ringraziamenti

Finalmente, dopo tutto questo tempo, raggiungo il mio obiettivo, non con pochi sacrifici. Potrà sembrare egoistico, ma per prima cosa vorrei ringraziare la mia forza di volontà e la mia perseveranza, che anche nei momenti peggiori, quando pensavo di dover mollare o di non farcela, mi hanno sempre spinta ad andare avanti per raggiungere questo obiettivo, primo (mi auguro!) di una lunga serie. La forza di volontà e la determinazione tuttavia, sono state corroborate da tutte le persone che ho accanto e che mi vogliono bene, e che mi hanno aiutata in questi anni.

Mia sorella Giusy, alla quale dedico questa tesi sotto sua pressione e minaccia verbale (scherzo, lo avrei fatto comunque), perché nonostante la lontananza c'è sempre stata per supportarmi, sopportarmi, e spronarmi ad andare avanti con la sua infinita serie di meme su di me e sulla mia carriera universitaria. Grazie, mi hai fatta ridere anche quando volevo solo piangere.

I miei genitori, che con il loro aiuto economico non indifferente mi hanno permesso e mi permettono ancora di frequentare l'università, e che sono stati comprensivi nonostante il mio comportamento a volte irragionevole.

Matteo, che in questi tre anni mi è sempre stato accanto sia come fidanzato che come collega. Grazie per aver condiviso con me sia giornate di studio matto e disperatissimo che momenti divertenti, se oggi ho raggiunto questo traguardo è anche grazie alle sue lezioni di matematica.

Il mio relatore, il prof. Salvatore Monteleone, che mi ha aiutata e accompagnata lungo questo percorso di tesi. Il nostro lavoro mi ha appassionata e mi hai spronata ad andare avanti per conto mio per approfondire gli argomenti, e forse un giorno sarò veramente in grado di realizzare progetti come questo. Chi lo sa, magari un giorno manderò davvero un rover su Marte.

A tutti i miei amici e colleghi, che anche se non li ho nominati uno per uno sanno di essere stati importanti ai fini del raggiungimento di questo mio traguardo, e che sono qui a festeggiare con me nonostante tutto.

Infine ultimi ma non meno importanti, ringrazio i miei nonni, che anche se purtroppo non hanno avuto il piacere di condividere direttamente questa giornata con me, mi stanno guardando da lassù con il cuore pieno di gioia, e so che anche loro stanno festeggiando. Vi voglio tanto bene.