



UNIVERSITÀ DEGLI STUDI DI FIRENZE
SCUOLA DI INGEGNERIA - DIPARTIMENTO DI INGEGNERIA
DELL'INFORMAZIONE

Tesi di Laurea Triennale in Ingegneria Informatica

**A WIRELESS IOT SYSTEM FOR ENVIRONMENTAL
VARIABLES SENSING, REMOTE DATA STORAGE
AND VISUALIZATION**

Candidato
Paula Mihalcea

Relatore
Prof. Andrew David Bagdanov

Correlatore
Walter Nunziati

Anno Accademico 2018 - 2019

A mia madre, che con me condivide gioie e dolori.
A Marco, che mi sostiene e incoraggia pazientemente.

Indice

1	Introduzione	1
1.1	Il progetto	3
1.2	Organizzazione della tesi	6
2	Connessione	8
2.1	Bluetooth Low Energy e protocollo GATT	8
2.2	Alcune specifiche: TI SimpleLink SensorTag CC2650STK	10
2.3	Alcune specifiche: Raspberry Pi 3 Model B+	11
2.4	Connessione al SensorTag	11
2.5	Acquisizione dati con gatttool	14
2.5.1	Esempio di acquisizione dati: temperatura	16
2.5.2	Alcuni handle di uso comune	17
3	Automazione	18
3.1	Interpretazione dei dati ricevuti	18
3.1.1	Temperatura	19
3.1.2	Umidità	20
3.1.3	Pressione	21
3.1.4	Luce	22
3.1.5	Movimento	23

3.2	Ciclo automatico di connessione ed acquisizione dati	29
3.2.1	Acquisizione delle impostazioni	30
3.2.2	Connessione al dispositivo	32
3.2.3	Configurazione dei sensori	33
3.2.4	Acquisizione periodica dei dati	33
3.2.5	Disconnessione	34
3.3	Connessioni multiple	34
3.4	Elaborazione dei dati	36
3.5	Plotting	37
4	Configurazione del server e database	40
4.1	Installazione di Elasticsearch	41
4.2	Popolamento del database	43
4.2.1	Alcuni comandi REST	43
4.3	Aggiornamento del database	44
4.3.1	Acquisizione delle impostazioni	45
4.3.2	Connessione al database	46
4.3.3	Inserimento log vecchi	46
4.3.4	Loop di aggiornamento	48
5	Visualizzazione dei dati	49
5.1	Installazione e configurazione di Kibana	50
5.1.1	Avvio e accesso	50
5.2	Visualizzazioni	51
5.3	Dashboard	54
5.4	Indici	55
6	Conclusioni	57
6.1	Range e batteria del SensorTag	57

6.2	Soluzioni concrete	58
6.3	Estensione del progetto	61
6.4	Considerazioni finali	61
Bibliografia		65
Ringraziamenti		69

Capitolo 1

Introduzione

IoT, ovvero **Internet of Things**: una parola che tutti, prima o poi, hanno sentito negli ultimi anni, e che continueranno a sentire con frequenza inevitabilmente maggiore. Cosa significa, però, questo termine? Spesso le definizioni che vengono fornite sono troppo vaghe, oppure troppo dettagliate, e sembra quasi che tutti sappiano di cosa si tratta ma nessuno sia in grado di delucidare davvero coloro che non hanno ancora le idee chiare. Prima di introdurre l'argomento di questa tesi, ovvero un progetto che di fatto concretizza un sistema IoT, è opportuno quindi spiegare tale concetto, giacché si trova alla base del prototipo che verrà estensivamente descritto nel resto dell'elaborato.

La parola “IoT” viene comunemente utilizzata per rappresentare un insieme di dispositivi interconnessi tramite Internet, solitamente inseriti in oggetti di uso comune e dotati di uno o più sensori, in grado di raccogliere ed inviare dati attraverso la rete. Tali dati vengono salvati e rielaborati per ricavarne informazioni utili, ed eventualmente utilizzati per inviare, in base ad essi, comandi agli stessi dispositivi che li hanno raccolti oppure ad attuatori di varia natura.

Un esempio concreto di sistema IoT potrebbe essere costituito, ad esempio, da alcuni sensori di temperatura installati nelle diverse stanze di una casa, che inviano periodicamente le loro letture ad un collettore connesso alla rete locale (ad es. un microcontrollore Raspberry). Tale dispositivo, a sua volta, inoltra i dati ricevuti ad un server collocato in casa (anche un semplice pc), che si occupa del loro salvataggio. Tramite un programma ad hoc, il server (oppure il collettore, se le operazioni da eseguire hanno bassa complessità) elabora periodicamente oppure in tempo reale i dati ricevuti e determina se, in base ad essi, è necessario attivare o disattivare l'impianto di condizionamento o altri apparecchi, secondo istruzioni precedentemente specificate dall'utente.

È possibile quindi individuare subito le **componenti fondamentali** di un sistema IoT, ovvero:

- uno o più **sensori**, con gli eventuali attuatori, connessi ad un solo collettore per volta;
- uno o più **collettori** connessi solitamente a più sensori/attuatori e in rete;
- un **server** collegato alla stessa rete;
- addizionalmente, nelle applicazioni più avanzate (come ad esempio quelle industriali), anche una o più **macchine dedicate** alla sola elaborazione dei dati, qualora il singolo server non sia più sufficiente per tale operazione.

L'utilità pratica dell'IoT risulta immediatamente evidente da questo semplice esempio, così come le enormi implicazioni che sistemi più complessi possono avere su larga scala. Questo aspetto non è certo sfuggito agli analisti ed

informatici di tutto il mondo, e sempre più aziende IT offrono ai loro clienti soluzioni IoT su misura e personalizzate. Non solo: innumerevoli dispositivi intelligenti (ovvero dotati di sensori e tecnologie di connessione wireless come il Bluetooth o il Wi-Fi) hanno fatto la loro comparsa sul mercato, a prezzi sempre più convenienti, al punto che si stima che nel 2020 ci saranno ben 20 miliardi di dispositivi connessi in rete [1], dei tipi più disparati.

Nello specifico, molti dei più recenti progetti IoT si rifanno alla necessità, all'interno di luoghi di lavoro particolarmente soggetti a rischio, di avere a disposizione strumenti in grado di **rilevare potenziali pericoli** ed **allertare gli operatori** in tempi utili per impedire danni a cose o persone, anche gravi. Nel mondo lavorativo esiste una vasta gamma di pericoli per la salute e la sicurezza delle persone e, nonostante l'avanzamento della tecnologia e le norme di sicurezza più stringenti, gli incidenti sono sempre in agguato, ed il giorno in cui saranno del tutto prevedibili ed evitabili rimane ancora molto lontano.

Il **settore edile** costituisce un particolare interesse per la ricerca in questo campo in quanto si caratterizza, dopo il settore agricolo, per la maggiore frequenza di infortuni (intesa come il rapporto tra incidenti denunciati e mille occupati). [2]

1.1 Il progetto

È proprio in questo contesto che si inserisce il lavoro qui presentato, svolto nell'azienda **Magenta s.r.l.** [3]. Si tratta di un prototipo per l'applicazione dell'IoT proprio nell'ambito della cantieristica edile, all'interno del progetto **Smartyard - SY 4.0** [4] e volta ad aumentare la sicurezza sul lavoro e l'efficienza gestionale secondo il paradigma dell'**Industria 4.0**, ovvero la

sempre più avanzata automazione dei processi produttivi attraverso l'interconnessione di dispositivi cosiddetti intelligenti. Lo scopo finale sarà quello di far evolvere il cantiere in una fabbrica intelligente [5] attraverso l'integrazione di varie discipline ingegneristiche impiegate per studiare, sviluppare ed utilizzare dispositivi e software innovativi, secondo anche quanto indicato nella decisione n. 20 del 11.04.2016 della Regione Toscana che stabilisce gli indirizzi per l'attuazione, appunto, dell'Industria 4.0 [6].

Il sistema IoT prodotto in seno a Magenta (fig. 1.1) costituisce quindi una base di partenza per il raggiungimento dell'obiettivo sopra introdotto, in quanto è composto da un insieme indipendente di dispositivi che, a partire da uno o più sensori BLE (Bluetooth Low Energy), utilizza un Raspberry Pi come accentratore ed elaboratore dati ed un pc (connesso alla stessa rete) per la loro archiviazione e visualizzazione, attraverso protocolli e piattaforme web standard.

Nello specifico, i dispositivi utilizzati sono i seguenti:

- **Texas Instruments SimpleLink SensorTag**, modello **CC2650STK**
- da qui in poi chiamato semplicemente SensorTag
- **Raspberry Pi 3 B+**, s.o. Raspbian GNU/Linux 9 (stretch)
- **Dell Latitude E5430**, s.o. Windows 10 Home x64

I programmi sviluppati sono stati scritti e testati con successo su queste macchine; tuttavia, gli script di per sé sono utilizzabili su qualunque oggetto in grado di comunicare tramite il protocollo BLE GATT, in quanto è sufficiente cambiare nei file di impostazioni i parametri del nuovo dispositivo (in particolare gli UUID dei handle per la configurazione e la lettura dei dati dei sensori) e fornire i corretti algoritmi per l'interpretazione dei dati ricevuti

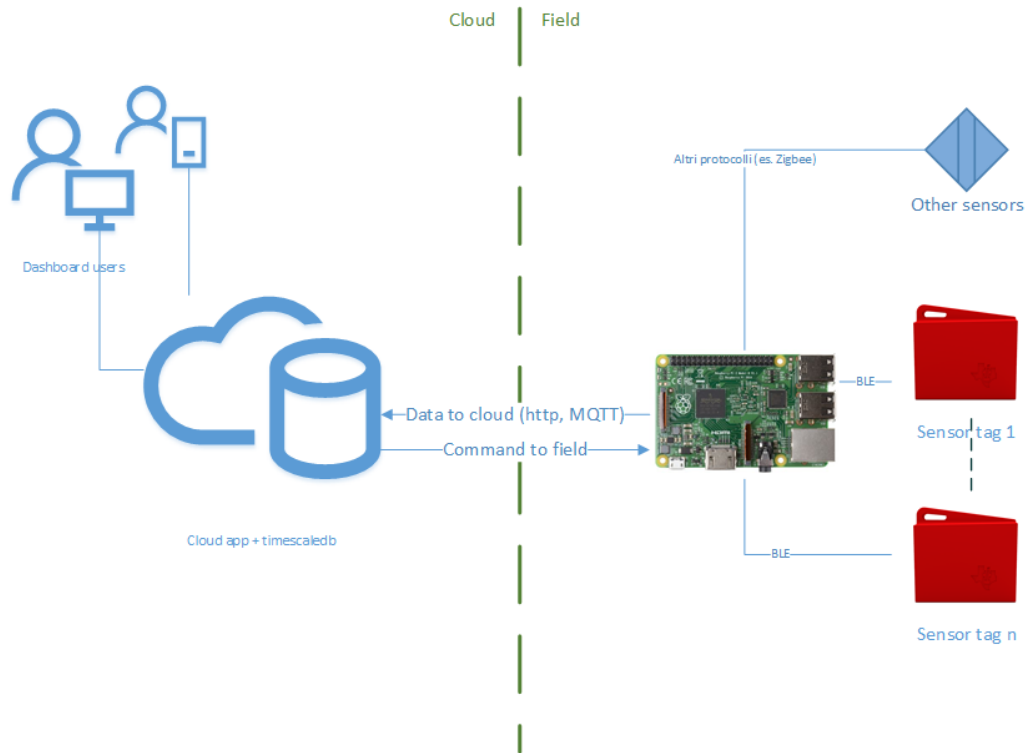


Figura 1.1: Struttura di alto livello del progetto.

(dipendenti dal dispositivo usato e tipicamente reperibili nel relativo data sheet), secondo le modalità descritte al capitolo 2.

Il prototipo corrente, seppur ben funzionante, non è comunque da considerarsi vicino al sistema finale. Il motivo risiede principalmente nel fatto che i dispositivi utilizzati non sono quelli effettivamente adatti all'uso previsto (in cantiere); inoltre, il programma non copre tutti i possibili casi d'uso, e dunque alcune funzionalità potrebbero risultare non sufficienti per un impiego nella vita reale. Tuttavia, il progetto dimostra come sia possibile realizzare una piccola rete di dispositivi IoT ed utilizzarla per misurare e monitorare efficacemente variabili d'ambiente rilevate ad intervalli regolari, abbastanza frequenti da risultare quasi in tempo reale: un'applicazione che, spostata in un contesto meno critico (ad esempio per controllare l'andamento di tem-

peratura e umidità nella propria abitazione), risulta pienamente valida ed efficiente.

1.2 Organizzazione della tesi

La tesi, che si propone come obiettivo la descrizione accurata dei passi seguiti nella realizzazione del prototipo in modo che il sistema finale sia facilmente riproducibile, sperabilmente anche da chi non possiede particolari conoscenze informatiche, è così organizzata:

- **Introduzione:** prima parte introduttiva, che si conclude con questo paragrafo;
- **Connessione:** breve descrizione delle tecnologie utilizzate e spiegazione del procedimento necessario per ottenere una connessione stabile tra SensorTag e Raspberry Pi, così come dei comandi utili per la ricezione dei dati acquisiti dai sensori (da linea di comando);
- **Automazione:** tecniche per l'interpretazione dei dati ricevuti dal SensorTag e descrizione dello script che realizza in modo automatico la connessione ed acquisizione periodica dei dati; il capitolo si conclude con alcune osservazioni sulla possibilità di connettere più dispositivi BLE contemporaneamente e con la spiegazione del funzionamento degli script che elaborano i dati salvati e di quelli che li plottano;
- **Configurazione del server e database:** descrizione del procedimento da seguire per l'installazione e la configurazione di Elasticsearch, ovvero del server che riceve e archivia i dati ricevuti dal Raspberry Pi;

- **Visualizzazione dei dati:** installazione e configurazione di Kibana, uno strumento legato ad Elasticsearch atto alla visualizzazione dei dati in forma grafica;
- **Conclusioni:** ultime osservazioni e conclusioni, oltre ad alcune linee guida per la scelta dei dispositivi hardware;
- **Bibliografia:** elenco delle fonti e guide citate nei vari capitoli;
- **Ringraziamenti:** perché una tesi, tutto sommato, non si scrive da soli.

Capitolo 2

Connessione

2.1 Bluetooth Low Energy e protocollo GATT

Il SensorTag, utilizzato per l'acquisizione dei dati attraverso i suoi sensori, fa uso della tecnologia **Bluetooth Low Energy** (BLE) [7] per trasmettere le informazioni acquisite all'accentratore. Tale protocollo, nato come variante del Bluetooth ed operante nella stessa banda (2,4 GHz, riservata a uso industriale, scientifico e medico), è nato per minimizzare il consumo energetico dei dispositivi che lo utilizzano (solitamente non molto potenti, e dalla breve autonomia) pur conservando l'affidabilità della connessione e mantenendo bassi i costi di produzione. È una tecnologia che grazie a queste caratteristiche sta riscontrando una diffusione sempre maggiore.

La standardizzazione del Bluetooth viene periodicamente revisionata dall'organizzazione Bluetooth Special Interest Group (SIG), che dal 1998 si occupa della definizione delle specifiche [8]. Come indicato a partire dalla versione 4.2 [9], il Bluetooth LE definisce due nuove core specifications (specifiche di base), **ATT** (Attribute Protocol) e **GATT** (Generic Attribute Protocol). La prima, di basso livello, descrive un protocollo per la scoperta, lettura e

scrittura dei dati su un dispositivo, reso possibile dalla definizione di diversi attributi, ovvero elementi composti da:

- un **handle** a 16 bit che definisce univocamente l'attributo;
 - un **UUID** che definisce il tipo dell'attributo;
 - un **valore** di una certa lunghezza, il cui significato dipende dall'UUID.
- [10]

Gli attributi sono elencati in una tabella e identificano, nell'insieme, ciò che un dispositivo può (e non può, se non c'è uno specifico handle) fare.

Il protocollo **GATT** consiste invece in un **framework** di livello superiore [11] che raggruppa attributi ATT in **caratteristiche** (characteristics), **servizi** (services) e **profili** (profiles). Ogni caratteristica è un attributo ben definito contenente un singolo valore logico; ad oggi ne sono state definite ufficialmente 230 per le applicazioni più disparate [12], da quelle inerenti la persona (misura della pressione sanguigna, peso, altezza, ecc.) a quelle legate al dispositivo (livello della batteria, versione firmware, ecc.). I servizi, invece, sono collezioni di caratteristiche e delle loro relazioni con gli altri servizi, e incapsulano il comportamento di una parte del dispositivo; similmente alle caratteristiche, ne esistono diversi predefiniti dallo standard corrente [13]. Infine i profili, che in realtà non esistono fisicamente nel dispositivo, sono insieme convenzionali di servizi, predefiniti dal produttore oppure dal Bluetooth SIG; la lista completa può essere consultata sul sito ufficiale del protocollo Bluetooth [11].

Per il corrente progetto, la core specification di maggior interesse è il GATT, in quanto punto di partenza per la connessione e l'acquisizione dei dati dai dispositivi Bluetooth LE.

2.2 Alcune specifiche: TI SimpleLink Sensor-Tag CC2650STK

Il TI SimpleLink SensorTag CC2650STK [14] è un microcontrollore dotato di un processore ARM Cortex M3 [15], alcuni sensori ed una batteria CR2032, in grado di acquisire dati ed inviarli tramite Bluetooth LE (oppure Zigbee o 6LoWPAN, abilitabili attraverso un aggiornamento del firmware), una volta stabilita una connessione. È stato scelto come prototipo perché si distingue da altri dispositivi simili per le dimensioni contenute (67 x 50 x 14 mm), il basso consumo energetico e la facilità di utilizzo. Il dispositivo infatti non richiede particolari conoscenze elettroniche o informatiche, ma si presenta pronto all'uso in quanto per accedere ai suoi sensori è sufficiente installare sul proprio smartphone l'applicazione dedicata del produttore per iOS oppure Android, e collegarsi tramite Bluetooth. I sensori montati a bordo del SensorTag sono i seguenti:

- **temperatura** (°C o °F, ambiente ed oggetto);
- **umidità** (%);
- **pressione** (hPa);
- **luce** (lx);
- **movimento** (giroscopio, accelerometro e magnetometro).

La Texas Instruments mette a disposizione diversi strumenti di sviluppo, che permettono di utilizzare il SensorTag in svariate applicazioni [16]. Nonostante ciò, nel presente lavoro si è preferito utilizzare un approccio di **reverse engineering** per accedere direttamente alle caratteristiche GATT

senza l'utilizzo di software proprietario, in modo che il codice rimanga indipendente dal produttore (ed anche dal dispositivo) e sia perciò utilizzabile in altri contesti.

2.3 Alcune specifiche: Raspberry Pi 3 Model B+

Il Raspberry Pi 3 Model B+ [17], come già accennato, è il dispositivo utilizzato come raccoglitore dati. È stato scelto per la sua potenza e versatilità: sebbene sia anch'esso un microcontrollore, monta un processore Cortex-A53 (ARM) @ 1,4GHz a 64 bit [18] ed ha una grande quantità di SDRAM a disposizione (1GB), oltre ad uno slot per micro SD cards che permette di avere una memoria flash arbitrariamente capiente per il sistema operativo, le applicazioni ed i dati. Inoltre, a differenza di altri dispositivi simili, questo Raspberry ha già montate a bordo diverse opzioni di connettività, fra cui il wireless LAN 2,4GHz/5GHz IEEE 802.11.b/g/n/ac, una porta Ethernet e, di particolare interesse, il Bluetooth 4.2 (BLE). Tali caratteristiche, assieme alla possibilità di collegare mouse e tastiera USB ed uno schermo HDMI, hanno reso il Raspberry il microcontrollore ideale per la realizzazione del prototipo Smartyard, in quanto riduce le difficoltà della programmazione host-target e facilita molto il debugging.

2.4 Connessione al SensorTag

Il primo passo per la realizzazione di un sistema IoT è senza ombra di dubbio la connessione tra l'accentratore ed i dispositivi dotati di sensori. Prima di cominciare è opportuno controllare che il SensorTag utilizzato per i

primi test funzioni correttamente, visualizzando su uno smartphone Android i suoi dati attraverso l'applicazione posta a disposizione dal produttore Texas Instruments.

Avendo quindi accertato che il microcontrollore opera regolarmente, si passa all'installazione sul Raspberry dello stack Bluetooth **Bluez** e delle librerie da cui esso dipende, ovvero `glib`, `d-bus`, `libudev`, `libical`, `readline`, `libusb`. Per questa operazione, ben documentata in rete [19], si utilizzano i seguenti comandi da terminale: [20]

```
1 sudo apt-get install libglib2.0-0 libglib2.0-dev
2 sudo apt-get install libdbus-1-dev
3 sudo apt-get install libudev-dev
4 sudo apt-get install libical-dev
5 sudo apt-get install libreadline-dev
6 sudo apt-get install libusb-1.0-0-dev
7 wget http://www.kernel.org/pub/linux/bluetooth/bluez-5.32.tar
   .xz
8 tar -xvf bluez-5.32.tar.xz
9 cd bluez-5.32
10 sudo ./configure --prefix=/usr --mandir=/usr/share/man --
    sysconfdir=/etc --localstatedir=/var --with-
    systemdsystemunitdir --with-systemduserunitdir --enable-
    library
11 sudo make
```

Rimane soltanto l'installazione di **gatttool** [21], uno strumento per sistemi Linux incluso con Bluez in grado di manipolare le caratteristiche Bluetooth LE GATT. Come menzionato prima, si vorrebbe **disaccoppiare** il codice dei programmi dal dispositivo e, soprattutto, dal produttore. Questo obiettivo viene reso possibile proprio da un accesso diretto alle caratteristiche BLE, senza alcuna applicazione proprietaria che serva da intermediario. Si

esegue quindi il seguente comando da terminale:

```
sudo cp attrib/gatttool /usr/local/bin
```

È dunque possibile, a questo punto, procedere con la connessione vera e propria al SensorTag. I primi test si eseguono dal terminale, con un metodo manuale atto a comprendere meglio il funzionamento di gatttool ed individuare le informazioni specifiche del SensorTag necessarie all'acquisizione dei dati. Per attivare il Bluetooth sul Raspberry ed avviare la ricerca dei dispositivi disponibili si scrive: [20]

```
1 sudo btmgmt
2 le on
3 exit
4 sudo hciconfig hci0 up
5 sudo hcitool lescan
```

Il SensorTag appare nell'elenco dei sistemi BLE con il seguente nome (dove la stringa iniziale rappresenta l'indirizzo MAC, unico per ogni dispositivo):

```
54:6C:0E:80:3F:01 CC2650 SensorTag
```

Se la lista non dovesse contenerlo, è sufficiente premere sul SensorTag il pulsante di accensione laterale per attivarlo; ciò potrebbe accadere perché il dispositivo **interrompe l'invio degli advertisements** dopo 120 secondi di inattività, allo scopo di limitare il proprio consumo energetico. Basta quindi copiare l'indirizzo MAC del dispositivo e, dopo aver premuto Ctrl+C per uscire dall'elenco dei dispositivi Bluetooth, eseguire il comando (sostituendo il suddetto indirizzo alle **xx**):

```
1 gatttool -b xx:xx:xx:xx:xx:xx --interactive
2 connect
```

Un messaggio di successo conferma l'avvenuta connessione al SensorTag, che rimane in attesa di ricevere ulteriori istruzioni in modalità interattiva.

2.5 Acquisizione dati con gatttool

L'acquisizione dei dati in modalità interattiva si è rivelata assai complessa e macchinosa, in quanto richiede una conoscenza approfondita dei handle utilizzati per ogni caratteristica messa a disposizione dal SensorTag (par. 2.5.2), e l'inserimento manuale di ogni istruzione da linea di comando. Tuttavia, è servita come base e valido stimolo per l'automatizzazione del processo di rilevazione delle variabili ambientali, perciò viene di seguito descritta in dettaglio.

Innanzitutto è necessario eseguire, una volta stabilita la connessione, il comando:

```
char-desc
```

il quale permette di ottenere l'elenco completo dei suddetti handle insieme agli UUID delle stesse caratteristiche. Ogni elemento nella lista ha il seguente formato¹, dove `UUUU` e `HHHH` indicano le parti dell'UUID di maggior interesse:

```
handle: 0xHHHH, uuid: f000UUUU-0451-4000-b000-000000000000
```

Questi dati sono necessari per accedere correttamente alle caratteristiche del SensorTag, perciò è opportuno salvarli in un file di testo per usi futuri. Si può dunque procedere alla configurazione dei sensori. Per poter ricevere dati, infatti, è necessario **attivare ogni sensore** di interesse tramite il comando:

```
char-write-cmd 0xHHHH 01
```

Se questo passaggio venisse omissso, un eventuale tentativo di accedere ai dati di un sensore non attivo restituisce 0 oppure l'ultimo dato rilevato prima della disattivazione.

¹I valori dei vari elementi costituenti ogni stringa potrebbero differire a seconda del dispositivo; tuttavia, ai fini del progetto sono necessarie soltanto le parti indicate con `HHHH` e `UUUU`.

Il comando:

```
char-read-hnd 0xHHHH
```

si utilizza invece per **ricevere i dati** relativi ad un particolare sensore.

Infine, dopo aver utilizzato un sensore, sarà possibile **disattivarlo** con:

```
char-write-cmd 0xHHHH 00
```

Ovviamente, in tutti i casi alla dicitura `HHHH` va sostituito il numero del handle della caratteristica desiderata; nel SensorTag ogni sensore ha da 1 a 4 handle diversi, generalmente destinati rispettivamente alla ricezione dati, attivazione delle notifiche, configurazione e impostazione del periodo di notifica. È possibile stabilire la giusta corrispondenza tra ogni handle ed il relativo UUID consultando le tabelle dei singoli sensori nella documentazione del dispositivo. Ciò significa che, una volta individuato nell'elenco ottenuto con `char-desc` l'UUID della caratteristica cui si è interessati (a sua volta ripreso dalle tabelle appena citate), si scriveranno, al posto di `HHHH` nei comandi di configurazione e ricezione dei dati, i rispettivi numeri di handle accanto al UUID presente nell'elenco ottenuto con `char-desc`.

Per **disconnettersi**, basta digitare da linea di comando:

```
disconnect
```

Seguono un esempio per chiarire meglio la procedura da eseguire ed un elenco dei handle di maggiore utilità per il SensorTag.

Nota: È interessante osservare che questo dispositivo mette a disposizione anche alcuni handle che permettono la ricezione periodica dei dati, una funzione pensata per evitare la ripetuta richiesta manuale con il metodo appena illustrato. Tuttavia, si è scelto di non fare uso di tale opzione, allo scopo di mantenere una maggiore flessibilità nella

configurazione del sistema da parte dell'utente (come ad esempio nella scelta del periodo).

2.5.1 Esempio di acquisizione dati: temperatura

Viene descritto di seguito un esempio di individuazione del handle per il sensore di temperatura a bordo del SensorTag, e del suo successivo utilizzo per l'acquisizione dei dati.

IR Temperature Sensor

The temperature sensor used on the SensorTag is Texas Instrument's TMP007 [\[4\]](#). The IR Temperature sensor is implemented with the following source code file:

Application: sensortag_tmp.c

Driver: SensorTmp007.c

Profile: irtempervice.c

NB! The TMP007 is no longer manufactured and is not fitted on SensorTags produced after June 2017.

IR Temperature Sensor				
Type	UUID	Access	Size (bytes)	Description
Data	AA01*	R/N	4	Object[0:7], Object[8:15], Ambience[0:7], Ambience[8:15]
Notification	2902	R/W	2	Write 0x0001 to enable notifications, 0x0000 to disable.
Configuration	AA02*	R/W	1	Write 0x01 to enable data collection, 0x00 to disable.
Period	AA03*	R/W	1	Resolution 10 ms. Range 300 ms (0x1E) to 2.55 sec (0xFF). Default 1 second (0x64)

Figura 2.1: Tabella del sensore di temperatura.

Dalla tabella relativa al sensore di temperatura, presente nella documentazione [22] e riportata nella 2.1 per semplicità, si evince che l'UUID necessario per attivare il sensore è il n°AA02 (*Configuration*). Consultando quindi l'elenco completo degli UUID, ottenuto precedentemente con il comando `char-desc`, si osserva che al UUID AA02 corrisponde il handle 0x0027: `handle: 0x0027, uuid: f000aa02-0451-4000-b000-000000000000`

Per attivare il sensore basta quindi scrivere da terminale:

```
char-write-cmd 0x0027 01
```

Il sensore inizia così a rilevare la temperatura, ed è quindi possibile interrogarlo in qualsiasi momento utilizzando il handle 0x0024, individuato allo

stesso modo grazie al UUID relativo ai dati, **AA01** (*Data*). Tale richiesta consiste nel comando:

```
char-read-hnd 0x0024
```

Il SensorTag ritorna a video il valore della temperatura rilevata in formato esadecimale, nella modalità descritta nella colonna *Description* della tabella. La sezione successiva della guida TI, ripresa nel paragrafo 3.1.1, spiega in dettaglio l'algoritmo necessario per elaborare ed interpretare correttamente i dati. Una volta ottenuti i dati desiderati, è possibile disattivare il sensore con:

```
char-write-cmd 0x0027 00
```

2.5.2 Alcuni handle di uso comune

Segue una breve lista dei handle di maggiore utilità del dispositivo utilizzato in questo progetto, un TI SimpleLink SensorTag CC2650STK.

Handle per la configurazione del sensore

```
Temperatura: 0x0027
Umidità: 0x002f
Barometro: 0x0037
Sensore ottico: 0x0047
Sensore di movimento: 0x003f
```

Handle per la richiesta dei dati

```
Temperatura: 0x0024
Umidità: 0x002c
Barometro: 0x0034
Sensore ottico: 0x0044
Sensore di movimento: 0x003c
```

Capitolo 3

Automazione

3.1 Interpretazione dei dati ricevuti

Sembrerebbe che, una volta stabilita la connessione al SensorTag e la modalità di acquisizione dei dati, si possa procedere all'automazione del processo. In realtà rimane ancora un passaggio da portare a termine prima di continuare, altrettanto importante delle operazioni precedentemente eseguite: **l'interpretazione dei dati**.

I dati che il SensorTag manda al dispositivo richiedente, infatti, non sono in un formato immediatamente comprensibile, bensì hanno una **codifica esadecimale little-endian**. È necessario quindi reperire dalla documentazione gli algoritmi necessari alla loro elaborazione e conversione in codifica decimale, ed eventualmente riscrivere le istruzioni lì presenti in un adeguato linguaggio di programmazione che esegua automaticamente queste operazioni.

Utilizzando dunque gli algoritmi originariamente in linguaggio C presenti alla sezione del server GATT della documentazione [22], sono state scritte le seguenti funzioni Python [23] per facilitare la lettura dei dati ricevuti;

verranno in seguito utilizzate per elaborare i dati prima di archivarli nel database del server.

3.1.1 Temperatura

```
1 def temp(raw_temp_data):
2     # raw_temp_data = 'c8 0a 68 0d' # Here's some test data
3     # received from the sensor
4     raw_temp_bytes = raw_temp_data.split() # Splits the data
5     # string into bytes
6
7     raw_ambient_temp = int('0x' + raw_temp_bytes[3] +
8     raw_temp_bytes[2], 16) # Conversion from hex to int
9     ambient_temp_int = raw_ambient_temp >> 2 # Right shift (
10    # as per TI algorithm)
11    ambient_temp_celsius = float(ambient_temp_int) * 0.03125
12    # Conversion to float and multiplication as per TI
13    # algorithm
14
15    # raw_obj_temp = int('0x' + raw_temp_bytes[1] +
16    raw_temp_bytes[0], 16) # Conversion from hex to int
17    obj_temp_int = raw_obj_temp >> 2 # Right shift (as per
18    # TI algorithm)
19    obj_temp_celsius = float(obj_temp_int) * 0.03125 #
20    # Conversion to float and multiplication as per TI
21    # algorithm
22
23    # print('Ambient temperature: ' + str(ambient_temp_celsius
24    # ) + '\degree C')
25    # print('Object temperature: ' + str(obj_temp_celsius) +
26    # '\degree C')
```



```
16     return ambient_temp_celsius
```

Questa funzione per la conversione della temperatura prende in ingresso una stringa esadecimale contenente le informazioni ricevute dal SensorTag e ritorna il valore della temperatura in **gradi centigradi** ($^{\circ}\text{C}$), secondo le istruzioni presenti nella documentazione; alternativamente può ritornarla in gradi Fahrenheit. [20] [22]

La **Ambient temperature** si riferisce alla temperatura dell'ambiente circostante, mentre la **Object temperature** a quella dell'oggetto su cui poggia il dispositivo.

3.1.2 Umidità

```
1 def hum(raw_hum_data):
2
3     # raw_hum_data = '84 62 c0 71' # Here's some test data
4     # received from the sensor
5     raw_hum_bytes = raw_hum_data.split() # Splits the data
6     # string into bytes
7
8     raw_hum = int('0x' + raw_hum_bytes[3] + raw_hum_bytes[2],
9                  16) # Conversion from hex to int
10    # raw_hum_temp = int('0x' + raw_hum_bytes[1] +
11    #                    raw_hum_bytes[0], 16) # Conversion from hex to int
12
13    hum = (float(raw_hum) / 65536) * 100 # Conversion to
14    # float and division as per TI algorithm
15    # hum_temp = (float(raw_hum_temp) / 65536) * 165 - 40 #
16    # Conversion to float and division as per TI algorithm
17
18    # print('Relative Humidity: ' + str(hum) + '%')
19    # print('Temperature (humidity sensor): ' + str(hum_temp)
20    #       + '\degree C')
```

```
14
15     return hum
```

La funzione di conversione dell'umidità prende in ingresso, come avviene per la temperatura, i dati grezzi ricevuti dal sensore, e ritorna in **percentuale** (%) il valore cercato, calcolato in base all'algoritmo presente nella documentazione. [22] È interessante notare come questo sensore sia in grado di rilevare anche la **temperatura ambientale**, memorizzata nei primi due bit; sperimentalmente si è visto che differisce di circa $\pm 0,4$ rispetto a quella rilevata del sensore di temperatura.

3.1.3 Pressione

```
1 def bar(raw_bar_data):
2
3     # raw_bar_data = '48 0a 00 d0 89 01' # Here's some test
4     # data received from the sensor
5     raw_bar_bytes = raw_bar_data.split() # Splits the data
6     # string into bytes
7
8     bar_int = int('0x' + raw_bar_bytes[5] + raw_bar_bytes[4] +
9                 raw_bar_bytes[3], 16) # Conversion from hex to int
10    # bar_temp_int = int('0x' + raw_bar_bytes[2] +
11    #                   raw_bar_bytes[1] + raw_bar_bytes[0], 16) # Conversion
12    # from hex to int
13
14    bar = float(bar_int) / 100 # Conversion to float and
15    # division as per TI algorithm
16
17    # bar_temp = float(bar_temp_int) / 100 # Conversion to
18    # float and division as per TI algorithm
19
20    # print('Barometric pressure: ' + str(bar) + ' hPa')
```

```
13     # print('Temperature (barometric pressure sensor): ' + str
        (bar_temp) + '\degree C')
14
15     return bar
```

Similmente, la funzione di conversione della pressione prende i dati grezzi ricevuti dal sensore e li rielabora (in realtà si tratta di una semplice conversione in formato decimale ed una divisione per 100), ritornando la pressione in **hPa**. [22] Anche questo sensore è in grado di rilevare la temperatura ambientale; sperimentalmente si è visto che questa differisce di circa $\pm 0,3$ rispetto a quella rilevata dal sensore di temperatura.

3.1.4 Luce

```
1 def opt(raw_opt_data):
2
3     # raw_opt_data = '59 8d' # Here's some test data received
        from the sensor
4     raw_opt_bytes = raw_opt_data.split() # Splits the data
        string into bytes
5
6     raw_opt = int('0x' + raw_opt_bytes[1] + raw_opt_bytes[0],
        16) # Conversion from hex to int
7
8     m = raw_opt & 0x0FFF # Masking as per TI algorithm
9     e = (raw_opt & 0x0FFF) >> 12 # Masking and right shift as
        per TI algorithm
10
11     if e == 0: # Operations as per TI algorithm
12         e = 1
13     else:
14         e = 2 << (e - 1)
15
```

```
16     opt = m * (0.01 * e) # More operations as per TI
    algorithm
17
18     # print('Light intensity: ' + str(opt) + ' lx')
19
20     return opt
```

I dati ricevuti dal sensore di intensità luminosa richiedono alcune operazioni dal significato non meglio specificato nell'algoritmo [22]; il risultato, in ogni caso, è il valore di luminosità espresso in **Lux**, lx.

3.1.5 Movimento

```
1 # Gyroscope
2 def gyro(raw_mov_data):
3     # Data bytes
4     # raw_mov_data = 'ad ff 27 01 00 00 00 02 4c 01 bc 3d 2f
    01 8f ff 53 fe' # Here's some test data received from
    the sensor
5     raw_mov_bytes = raw_mov_data.split() # Splits the data
    string into bytes
6
7     # x axis
8     gyro_x_int = int('0x' + raw_mov_bytes[1] + raw_mov_bytes
    [0], 16) # Conversion from hex to int
9     gyro_x = float(gyro_x_int) / (65536 / 500) # Conversion
    to float and operations as per TI algorithm
10
11     # y axis
12     gyro_y_int = int('0x' + raw_mov_bytes[3] + raw_mov_bytes
    [2], 16) # Conversion from hex to int
13     gyro_y = float(gyro_y_int) / (65536 / 500) # Conversion
    to float and operations as per TI algorithm
14
```

```
15     # z axis
16     gyro_z_int = int('0x' + raw_mov_bytes[5] + raw_mov_bytes
17                       [4], 16) # Conversion from hex to int
18     gyro_z = float(gyro_z_int) / (65536 / 500) # Conversion
19     to float and operations as per TI algorithm
20
21     # print('Gyroscope      x: ' + str(gyro_x) + ' deg/s    y:
22     ' + str(gyro_y) + ' deg/s    z: ' + str(gyro_z) + ' deg/
23     s')
24
25     return gyro_x, gyro_y, gyro_z
26
27
28
29
30 # Accelerometer
31 def acc(raw_mov_data):
32     # Data bytes
33     # raw_mov_data = 'ad ff 27 01 00 00 00 02 4c 01 bc 3d 2f
34     01 8f ff 53 fe' # Here's some test data received from
35     the sensor
36     raw_mov_bytes = raw_mov_data.split() # Splits the data
37     string into bytes
38
39     # Configuration bytes
40     raw_config_data = '7f 03' # Here's the current Config
41     value of the sensor
42     raw_config_bytes = raw_config_data.split() # Splits the
43     data string into bytes
44
45     config_bytes = int('0x' + raw_config_bytes[1], 16) #
46     Conversion from hex to int
47     config_bytes_bin = list(bin(config_bytes)[2:].zfill(
48     8)) # Conversion from int to binary (every bit sets
49     on or off a certain feature, so binary is easier to
```

```
        work with - see sensor Config tables)

37
38     def acc_range(acc_a): # Function needed to determine the
        necessary operation to process the data, as per TI
        algorithm
39         acc_range_bit = int(config_bytes_bin[7] +
            config_bytes_bin[6],
40                                2) # Conversion from binary to
                                    int (to get the corresponding
                                    range value)

41
42         if acc_range_bit == 0:
43             r = 2
44         elif acc_range_bit == 1:
45             r = 4
46         elif acc_range_bit == 2:
47             r = 8
48         elif acc_range_bit == 3:
49             r = 16
50
51         v = acc_a / (32768 / r)
52         return v
53
54     # x axis
55     acc_x_f = float(int('0x' + raw_mov_bytes[7] +
        raw_mov_bytes[6], 16)) # Conversion from hex to int,
        then float
56     acc_x = acc_range(acc_x_f)
57
58     # y axis
59     acc_y_f = float(int('0x' + raw_mov_bytes[9] +
        raw_mov_bytes[8], 16)) # Conversion from hex to int,
        then float
```

```
60     acc_y = acc_range(acc_y_f)
61
62     # z axis
63     acc_z_f = float(int('0x' + raw_mov_bytes[11] +
        raw_mov_bytes[10], 16)) # Conversion from hex to int,
        then float
64     acc_z = acc_range(acc_z_f)
65
66     # print('Accelerometer  x: ' + str(acc_x) + ' G   y: ' +
        str(acc_y) + ' G   z: ' + str(acc_z) + ' G')
67
68     return acc_x, acc_y, acc_z
69
70
71 # Magnetometer
72 def mag(raw_mov_data):
73     # Data bytes
74     # raw_mov_data = 'ad ff 27 01 00 00 00 02 4c 01 bc 3d 2f
        01 8f ff 53 fe' # Here's some test data received from
        the sensor
75     raw_mov_bytes = raw_mov_data.split() # Splits the data
        string into bytes
76
77     # x axis
78     mag_x = float(int('0x' + raw_mov_bytes[13] + raw_mov_bytes
        [12], 16)) # Conversion from hex to int, then float
79
80     # y axis
81     mag_y = float(int('0x' + raw_mov_bytes[15] + raw_mov_bytes
        [14], 16)) # Conversion from hex to int, then float
82
83     # z axis
84     mag_z = float(int('0x' + raw_mov_bytes[17] + raw_mov_bytes
```

```
[16], 16)) # Conversion from hex to int, then float
85
86 # print('Magnetometer x: ' + str(mag_x) + ' uT y: ' +
      str(mag_y) + ' uT z: ' + str(mag_z) + ' uT')
87
88 return mag_x, mag_y, mag_z
```

Il sensore di movimento incorpora ben tre sensori differenti: **giroscopio**, **accelerometro** e **magnetometro**. Come descritto nella documentazione fornita dal produttore, i dati di tutti e tre i sensori vengono forniti in un'unica stringa di 18 byte, che vanno quindi suddivisi per ogni sensore e, ulteriormente, per ogni asse (x, y e z). [22]

Nei due byte di configurazione del sensore è possibile abilitare e disabilitare a piacere, come specificato nella tabella 3.1, ogni asse del giroscopio e accelerometro, oltre al magnetometro (per quest'ultimo soltanto tutti e tre gli assi insieme). Inoltre, questo sensore consente di attivare o disattivare una funzionalità particolare detta **Wake-On-Motion** (WOM) che permette al sensore di movimento di utilizzare solamente l'accelerometro ed attivare tramite un'interruzione al SensorTag, per 10 secondi, anche giroscopio e magnetometro, solo quando rileva un movimento più forte (ad es. agitando il SensorTag). Gli ultimi due bit di configurazione invece indicano il **range** dell'accelerometro, compreso tra 0 e 3 e pari a, rispettivamente, 2G, 4G, 8G e 16G.

La tabella 3.1, basata sulla documentazione ma comunque parzialmente ricavata sperimentalmente, può essere utilizzata come riferimento per scrivere la corretta configurazione (ogni bit sarà a 0 o 1 a seconda se il rispettivo sensore dovrà essere spento o acceso). La stringa di 0 e 1 così ottenuta non è tuttavia adatta ad essere inviata al SensorTag con il comando char-write-cmd; prima di venire utilizzata deve essere **convertita** in un numero esadecimale

a quattro cifre (con un qualsiasi tool facilmente reperibile in rete).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
WOM 1	Mag (x, y, z)	Acc x	Acc y	Acc z	Gyro x	Gyro y	Gyro z	X	X	X	X	X	X	Range (LSB)	Range (MSB)

Tabella 3.1: Bit di configurazione del sensore di movimento.

Segue un esempio di configurazione del sensore di movimento per meglio illustrare l'utilizzo della tabella.

Esempio di configurazione

Supponiamo di voler impostare il sensore di movimento con la funzionalità WOM, gli assi x e z del giroscopio e l'asse y dell'accelerometro disattivati (mentre tutti gli altri sensori saranno attivi), e range pari a 2. Utilizzando la tabella qui sopra, la stringa di bit da codificare è costituita nel seguente modo: 0110101000000001.

La codifica esadecimale di tale stringa è quindi 6a01 e, di conseguenza, il comando da inviare al SensorTag è `char-write-cmd 0x003c 6a01`.

Nella tabella 3.2 si riportano, per convenienza, alcuni codici esadecimali di configurazione, corrispondenti alle varie combinazioni di WOM on/off e dei quattro range dell'accelerometro (0, 1, 2 e 3) con tutti i sensori di movimento attivi.

Range	WOM on	WOM off
0	ff 00	7f 00
1	ff 02	7f 02
2	ff 01	7f 01
3	ff 03	7f 03

Tabella 3.2: Alcuni codici esadecimali di configurazione del sensore di movimento.

Nota: Si è osservato che il comando di scrittura al handle `3f`, corrispondente alla configurazione del sensore di movimento, funziona soltanto la prima volta che viene eseguito (dopo la connessione al SensorTag). Ulteriori tentativi di modifica porteranno all'impostazione `ff ff`, corrispondente alla stringa `11111111 11111111`, che non sarà più possibile modificare (sospetto bug). La configurazione potrà quindi essere sovrascritta soltanto dopo aver disconnesso e connesso nuovamente il SensorTag; si è osservato anche che dopo questa operazione la configurazione ritorna allo stato di default, `00 02`.

3.2 Ciclo automatico di connessione ed acquisizione dati

Il processo di connessione al SensorTag e di acquisizione dati su Linux può essere automatizzato con uno script Python grazie alla libreria **Pexpect** [24], che permette di aprire un processo interattivo da terminale ed inviargli comandi come se fossero scritti da un utente in carne ed ossa.

Dopo il recupero iniziale dei parametri necessari, letti dal file `setup.ini` (situato nella cartella `config`), il programma procede alla connessione e all'acquisizione dei dati. Pexpect, naturalmente, viene utilizzata per controllare gatttool.

Lo script **connect.py**, reperibile alla repository GitHub del progetto [25], esegue quindi queste operazioni in cinque fasi distinte:

- acquisizione delle impostazioni;
- connessione al dispositivo;
- configurazione dei sensori;

- acquisizione periodica dei dati;
- disconnessione.

È possibile avviare questo programma da linea di comando, digitando nella cartella principale del progetto

```
python3 connect.py 1
```

Il parametro da linea di comando sta ad indicare il numero del file di setup che il programma dovrà utilizzare (vedi 3.3).

3.2.1 Acquisizione delle impostazioni

L'acquisizione delle impostazioni avviene tramite la libreria Python **ConfigParser** [26], che legge il file **setup.ini** (nella cartella `config`). Esso contiene, in formato INI, tutte le possibili variabili dello script, realizzando il completo disaccoppiamento dei dati dal programma, il quale risulta di conseguenza utilizzabile con qualsiasi dispositivo BLE. In `setup.ini`, infatti, ci sono tutti i tempi di attesa necessari ad una corretta connessione, i handle per la configurazione dei sensori e la ricezione dei dati, i valori di configurazione, la lunghezza dei dati ricevuti e la cartella di salvataggio dei log. Ovviamente, lo stesso file contiene anche l'indirizzo MAC del dispositivo a cui ci si vuole collegare. Di seguito vengono elencati e spiegati in dettaglio tutti i parametri.

Sezioni del file `setup.ini`

`[device_mac]`: è l'indirizzo MAC del dispositivo BLE

`[wait_time]`: vari tempi di attesa; sono tutti in millisecondi

`connection_timeout`: tempo di attesa per la connessione al dispositivo; il programma ritorna un errore ed esce automaticamente se questo tempo viene superato

`data_timeout`: tempo di attesa per la ricezione dei dati; si applica ad ogni sensore interrogato, ed è simile al `connection_timeout`

`wait_after_config`: tempo di attesa tra l'avvenuta configurazione dei sensori e l'inizio del ciclo di acquisizione dati; si consiglia di impostarlo a qualche secondo, in modo che alla prima lettura dei dati i sensori siano attivi e non diano risultati nulli o di default

`time_between_sensor_configs`: tempo di attesa tra la configurazione di ogni sensore; serve per evitare conflitti tra comandi e/o il flooding del dispositivo BLE

`time_between_data_requests`: tempo di attesa tra le letture dei sensori, all'interno dello stesso ciclo di lettura; ha lo stesso scopo di `time_between_sensor_configs`

`period`: periodo di lettura dei dati; a scelta dell'utente

`max_attempts`: numero massimo consentito di tentativi per la connessione iniziale e, in caso di perdita del segnale, la riconnessione

`[logs_path]`: nome della sottocartella di salvataggio dei log; deve trovarsi nella stessa cartella del programma

```
principale, altrimenti va specificato l'intero path

[config_handles]: handle per la configurazione dei sensori

[config_values]: valori per la configurazione dei sensori;
tipicamente saranno 00 per un sensore spento e 01 per uno
attivo

[data_handles]: handle per la lettura dei dati

[data_length]: lunghezza della stringa ricevuta dal
dispositivo BLE, ovvero il numero di caratteri ricevuti,
inclusi gli spazi (NON il numero totale di byte);
necessaria per leggere correttamente i valori ritornati
dai sensori
```

3.2.2 Connessione al dispositivo

La connessione, come menzionato in precedenza, avviene grazie a Pexpect. Lo script infatti avvia gatttool [27], invia la richiesta di connessione al dispositivo specificato in `setup.ini` ed attende la risposta `Connection successful`, che conferma l'avvenuta connessione. In caso di raggiungimento del tempo massimo di attesa (`connection_timeout`), il programma ritenta la connessione per il numero di volte massimo specificato nel file `setup.ini` (`max_attempts`) dopodiché, se non ha successo, si chiude automaticamente.

Il programma procede quindi inviando al database la notifica dell'avvenuta connessione, attraverso la funzione `modules/notify.py`.

3.2.3 Configurazione dei sensori

Per ogni sensore presente nel file `setup.ini`, il programma invia al dispositivo la richiesta di configurazione con il rispettivo valore (specificato nello stesso file), con un breve tempo di attesa tra una richiesta ed un'altra (`time_between_sensor_configs`) allo scopo di evitare che ne riceva troppe insieme.

3.2.4 Acquisizione periodica dei dati

A questo punto il programma entra in un **ciclo infinito di lettura dei dati**, eseguendo periodicamente un sottociclo di richiesta e lettura dei dati per ogni sensore tramite `gatttool` e `Pexpect`. Similmente alla fase di configurazione, c'è un tempo di attesa tra la lettura di un sensore e la successiva (`time_between_data_requests`).

Lo script procede quindi al **salvataggio** dei dati ricevuti (senza averli elaborati) su un file di testo avente per nome la data corrente (formato `YYYY-MM-dd.log`, ad es. `2019-07-15.log`) in un formato tabellare simile ad un dizionario Python. Ogni riga contiene la data e l'ora dell'acquisizione in formato ISO 8601 [28], l'indirizzo MAC del dispositivo ed una serie di coppie chiave-valore, corrispondenti rispettivamente al sensore ed al valore misurato. Tale formato è stato pensato per ottimizzare la lettura e l'elaborazione dei dati, facilmente eseguibile con la funzione `modules/raw2df.py` (che ritorna un dataframe Pandas, vedi par. 3.4), pur mantenendo la leggibilità degli stessi nei log originali.

3.2.5 Disconnessione

L'utente può interrompere in qualunque momento l'acquisizione dei dati premendo `Ctrl+C`; dopo aver catturato l'eccezione di interruzione da tastiera, il programma provvede automaticamente a disconnettere il dispositivo con il comando `gatttool disconnect`.

In caso di perdita della connessione, lo script **tenta automaticamente la riconnessione** per il numero massimo di volte consentito, dopodiché chiude il programma.

Infine, a prescindere dal motivo della disconnessione, prima di chiudersi il programma invia al database la notifica dell'avvenuta disconnessione, sempre attraverso la funzione `modules/notify.py`.

3.3 Connessioni multiple

In seguito all'automatizzazione del processo di connessione è stata testata con successo la possibilità di connettere il Raspberry a **due** oppure **tre** **SensorTag contemporaneamente** (vedi fig. 6.3).

La procedura consiste semplicemente nell'**avviare più script connect.py** da **terminali diversi**, specificando da linea di comando il **numero del file di impostazioni** che l'istanza dovrà utilizzare (1 per quello di default, che non ha alcun numero nel proprio nome).

Prima di avviare i `connect.py` bisogna fare nella cartella `config` tante copie di `setup.ini` e `setup_db.ini` quanti sono i dispositivi, denominandole `setup2.ini`, `setup3.ini`, ecc., ed avere l'accortezza di modificare al loro interno l'indirizzo MAC dei nuovi SensorTag (recuperabile con una scansione dei dispositivi BLE, `sudo hcitool lescan`, vedi par. 2.4) e la cartella dei log. Ogni dispositivo necessita infatti anche di una cartella dei log separata.

Per la connessione contemporanea di tre SensorTag, ad esempio, è sufficiente scrivere i seguenti comandi in tre terminali distinti:

```
python3 connect.py 1 # Si riferisce ai file di impostazioni
                      setup.ini e setup_db.ini; default (va comunque sempre
                      specificato, anche quando c'è un unico dispositivo)
```

```
python3 connect.py 2 # Si riferisce ai file di impostazioni
                      setup2.ini e setup_db2.ini
```

```
python3 connect.py 3 # Si riferisce ai file di impostazioni
                      setup3.ini e setup_db3.ini
```

All'interno della cartella principale del programma (Smartyard-SY4.0) la configurazione di file e directory per l'esempio a tre dispositivi è così fatta:

```
- Smartyard-SY4.0
  - config
    * last.date
    * setup.ini
    * setup2.ini
    * setup3.ini
    * setup_db.ini
    * setup_db2.ini
    * setup_db3.ini
  - logs
  - logs2
  - logs3
  - modules
  - sensors
  * connect.py
  * plot.py
  * plot_double.py
  * process_data.py
```


In mancanza di altri dispositivi oltre ai tre già citati, non è stato possibile determinare il numero massimo di SensorTag che possono essere connessi contemporaneamente al Raspberry Pi. Non essendo un simile limite specificato nella documentazione BLE, l'unico modo per trovarlo sarebbe quello empirico; tuttavia, ciò richiederebbe un numero arbitrariamente elevato di SensorTag a disposizione, un requisito che esula dallo scopo della tesi e sicuramente anche dalla disponibilità economica concessa a quello che, al momento, è soltanto un prototipo.

3.4 Elaborazione dei dati

I dati raccolti nei file di log (presenti nella sottocartella **logs** con formato `.log` e nome assegnato secondo la data ed il giorno in cui sono stati acquisiti i dati) possono essere elaborati attraverso lo script **raw2df.py** [25], presente nella sottocartella `modules` e implementato sotto forma di una funzione che prende come parametro il percorso del log da cui estrarre i dati.

All'avvio questo script carica automaticamente dalla cartella **sensors**, precedentemente popolata con gli algoritmi di interpretazione dei dati grezzi¹, i moduli dei sensori a disposizione, leggendo il loro nome dal file `setup.ini`. Conoscendo il formato in cui sono stati salvati i dati dei log, il programma procede al loro caricamento e trasformazione in un dizionario Python.

Lo script conclude quindi le sue operazioni con la creazione di un **data-frame** nel formato della libreria **Pandas** [29], il quale viene ritornato alla funzione chiamante e può essere utilizzato in seguito da altri programmi.

¹`sensors` contiene le funzioni per l'elaborazione dei dati descritte al par. 3.1, salvate singolarmente in tanti file quanti sono i handle del dispositivo, ed i cui nomi coincidono con quelli assegnati agli stessi handle nel file `setup.ini`.

Nel caso del SensorTag, la cartella `sensors` contiene gli algoritmi TI trascritti in Python per l'interpretazione dei dati ricevuti dai sensori di temperatura, umidità, pressione, intensità luminosa e movimento, come descritto nella relativa pagina della documentazione [22]. Da notare che al momento i dati del sensore di movimento non vengono elaborati, ma sono scartati dal dataframe finale, in quanto l'obiettivo del progetto è quello di visualizzare le variazioni delle variabili d'ambiente nel tempo, ed i dati di movimento non risultano di particolare interesse per questo scopo. Eventuali sviluppi futuri potrebbero includere un adeguato utilizzo anche dei dati di movimento raccolti.

Ovviamente lo script appena descritto è utile soltanto per trasformare singoli log in dataframe Pandas; ulteriori programmi per leggere tutti i file di log disponibili e manipolarli in modi più complessi verranno esposti in seguito.

3.5 Plotting

Sebbene la visualizzazione dei dati venga trattata in modo approfondito al capitolo 5, si presenta di seguito anche un semplice strumento di **plotting Python** che si è rivelato molto utile durante la fase di debug del programma.

Lo script `plot.py` [25] per la visualizzazione grafica dei dati raccolti, disponibile nella repository del progetto, utilizza il dataframe creato da `raw2df.py` per indicizzare i dati di un singolo log (specificato all'interno del programma) rispetto alla data ed ora a cui sono stati rilevati, e creare alcuni grafici esplicativi tramite la libreria **pyplot** [30].

Nel caso del SensorTag, lo script visualizza temperatura ambientale, umidità, pressione e intensità luminosa (ignorando il sensore di movimento, i cui

dati non avrebbero molto significato in un grafico bidimensionale). L'immagine generata può essere salvata in diversi formati e riutilizzata in seguito secondo l'esigenza, come evidente dalla figura 3.1.

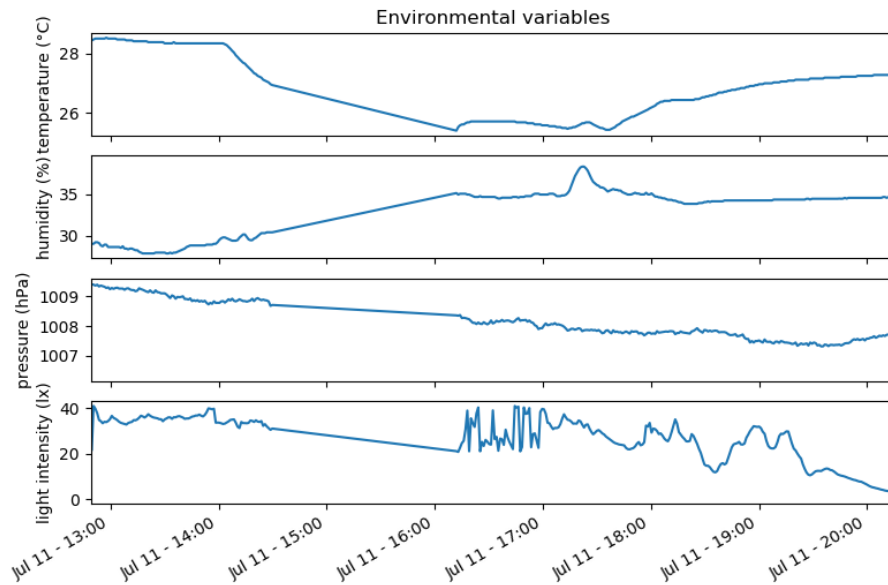
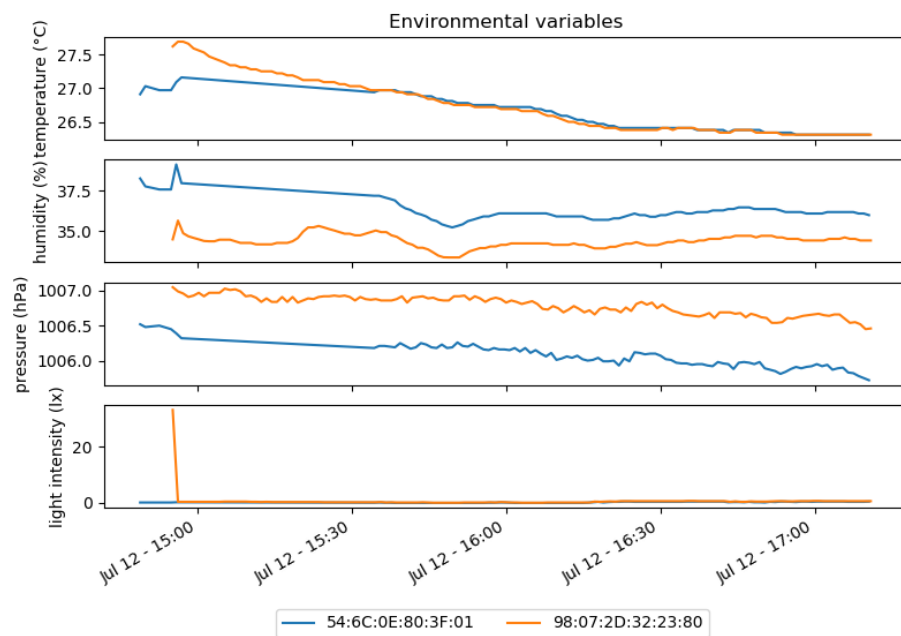


Figura 3.1: Grafico generato da `plot.py`.

In aggiunta a questo programma, lo script `plot_double.py` [25] permette di visualizzare contemporaneamente i dati di due sensori, ottenuti semplicemente caricando due file di dati diversi (anch'essi specificati all'interno del codice) (3.2).

Come si evince dalle figure 3.1 e 3.2, il risultato è graficamente semplice e pulito. Ai fini del debug, soprattutto per capire se i dati raccolti sono consistenti con le variazioni ambientali effettivamente osservate durante il periodo di acquisizione, questi programmi si sono rivelati assai funzionali.

Figura 3.2: Grafico generato da `plot_double.py`.

Capitolo 4

Configurazione del server e database

Elasticsearch [31] è un motore di ricerca open core basato sulla libreria Apache Lucene [32]. Si tratta di un server e database **RESTful** [33] distribuito, pensato per essere scalabile e molto veloce; orientato ai documenti, è in grado di fornire risultati in tempo quasi reale. Elasticsearch è **senza schema**: questo significa che per indicizzare un qualunque documento è sufficiente specificare l'**indice** in cui archiviarlo (ovvero il database, dato che ce ne possono essere diversi sullo stesso server); se non viene specificato, l'ID della nuova istanza viene generato automaticamente in modo univoco.

Grazie a queste caratteristiche, ed al fatto di essere gratuito, Elasticsearch ha conosciuto una **grande diffusione** negli ultimi anni, e diverse note compagnie lo usano per gestire i propri dati, come ad esempio Netflix, Stack Overflow, LinkedIn e Medium [34].

La scelta del server di ricerca da utilizzare nel progetto della tesi è quindi naturalmente caduta su questo strumento, sia per la sua facilità di installazione in ambienti Linux e Windows, sia per il suo valore didattico e

commerciale.

Inoltre, Elasticsearch risulta anche particolarmente adatto ad essere integrato con Python, in quanto accetta documenti in formato JSON e possiede una libreria apposita per consentire agli script l'interazione diretta col database.

4.1 Installazione di Elasticsearch

Per installare Elasticsearch su Windows, il sistema operativo della macchina usata come server (vedi 1.1), è sufficiente scaricare l'ultima versione dal sito `elastic.co` [31] ed estrarla in una qualunque posizione sul disco fisso; per configurare il server in modo che sia accessibile tramite chiamate REST (vedi 4.2) dalle macchine nella rete locale (LAN) basta aprire il file:

```
/elasticsearch-7.2.0/config/elasticsearch.yml
```

per modificare alcune impostazioni di base, come segue:

```
node.name: node-1    # Sezione Node
network.host: 0.0.0.0 # Sezione Network
discovery.seed_hosts: ["127.0.0.1", "[:,1]"] # Sezione
Discovery
cluster.initial_master_nodes: node-1 # Sezione Discovery
```

È fondamentale ricordarsi, prima di salvare il file così modificato, di **rimuovere il commento** # presente di default davanti alle quattro nuove impostazioni, che altrimenti non avranno alcun effetto.

A questo punto per avviare il server basta eseguire il file `/elasticsearch-7.2.0/bin/elasticsearch.bat`, e **lasciare il processo esecuzione** (fig. 4.2). È possibile verificare se Elasticsearch sta funzionando correttamente visitando l'indirizzo `http://localhost:9200` da un qualunque browser sulla mac-

```

[2019-08-01T23:11:50,720][INFO ][o.e.x.m.p.l.CppLogMessageHandler] [node-1] [controller/11328] [Main.cc@110] controller
(64 bit): Version 7.2.0 (Build 65aefcbfce449b) Copyright (c) 2019 Elasticsearch BV
[2019-08-01T23:11:51,018][DEBUG][o.e.a.ActionModule] [node-1] Using REST wrapper from plugin org.elasticsearch.xp
ack.security.Security
[2019-08-01T23:11:51,285][INFO ][o.e.d.DiscoveryModule] [node-1] using discovery type [zen] and seed hosts providers
[settings]
[2019-08-01T23:11:51,798][INFO ][o.e.n.Node] [node-1] initialized
[2019-08-01T23:11:51,799][INFO ][o.e.n.Node] [node-1] starting ...
[2019-08-01T23:11:52,271][INFO ][o.e.t.TransportService] [node-1] publish_address {192.168.1.2:9300}, bound_addresses
{:::9300}
[2019-08-01T23:11:52,275][INFO ][o.e.b.BootstrapChecks] [node-1] bound or publishing to a non-loopback address, info
rming bootstrap checks
[2019-08-01T23:11:52,301][INFO ][o.e.c.c.Coordinator] [node-1] cluster UUID [S21SK57oSpezjDwPyRb1ng]
[2019-08-01T23:11:52,393][INFO ][o.e.c.s.MasterService] [node-1] elected-as-master ([1] nodes joined)[{node-1}{3Allh
B_SSwyzEemurkjVnQ}{8LkwxoOYSbCSx3YggaoOjQ}{192.168.1.2}{192.168.1.2:9300}{ml.machine_memory=17123532800, xpack.installed
=true, ml.max_open_jobs=20} elect leader, _BECOME_MASTER_TASK, _FINISH_ELECTION_], term: 2, version: 47, reason: master
node changed {previous [], current [{node-1}{3AllhB_SSwyzEemurkjVnQ}{8LkwxoOYSbCSx3YggaoOjQ}{192.168.1.2}{192.168.1.2:9
300}{ml.machine_memory=17123532800, xpack.installed=true, ml.max_open_jobs=20}]}, term: 2, version: 47, reason: Publication{term=2, version=47}
[2019-08-01T23:11:52,546][INFO ][o.e.c.s.ClusterApplierService] [node-1] master node changed {previous [], current [{nod
e-1}{3AllhB_SSwyzEemurkjVnQ}{8LkwxoOYSbCSx3YggaoOjQ}{192.168.1.2}{192.168.1.2:9300}{ml.machine_memory=17123532800, xpack
.installed=true, ml.max_open_jobs=20}]}, term: 2, version: 47, reason: Publication{term=2, version=47}
[2019-08-01T23:11:52,768][INFO ][o.e.h.AbstractHttpServerTransport] [node-1] publish_address {192.168.1.2:9200}, bound_a
ddresses {:::9200}
[2019-08-01T23:11:52,768][INFO ][o.e.n.Node] [node-1] started
[2019-08-01T23:11:52,777][INFO ][o.e.l.LicenseService] [node-1] license [fac43b35-1d6b-432b-9b47-26dc19efbd00] mode
[basic] - valid
[2019-08-01T23:11:52,784][INFO ][o.e.g.GatewayService] [node-1] recovered [4] indices into cluster_state
[2019-08-01T23:11:55,917][INFO ][o.e.c.r.a.AllocationService] [node-1] Cluster health status changed from [RED] to [YELL
OW] (reason: [shards started [[sensortag][0]] ...]).

```

Figura 4.1: Processo `elasticsearch.bat` a server avviato ed in funzione.

china del server, oppure da un'altra macchina nella rete locale sostituendo l'indirizzo IP del server a `localhost`.

```

{
  "name": "DESKTOP-HU2HPEN",
  "cluster_name": "elasticsearch",
  "cluster_uuid": "xu919qxTRvWjuBDBk7j40w",
  "version": {
    "number": "7.2.0",
    "build_flavor": "default",
    "build_type": "zip",
    "build_hash": "508c38a",
    "build_date": "2019-06-20T15:54:18.811730Z",
    "build_snapshot": false,
    "lucene_version": "8.0.0",
    "minimum_wire_compatibility_version": "6.8.0",
    "minimum_index_compatibility_version": "6.0.0-beta1"
  },
  "tagline": "You Know, for Search"
}

```

Figura 4.2: Pagina accessibile dall'indirizzo `http://localhost:9200`, che testimonia il corretto funzionamento del server Elasticsearch.

Il client, un Raspberry Pi con Linux nel caso del progetto corrente, non avrà bisogno di installare Elasticsearch, ma potrà contattare il server direttamente da linea di comando tramite lo strumento **curl** [35].

4.2 Popolamento del database

Come già accennato, il server avviato nella sezione precedente può essere acceduto da qualunque macchina nella rete LAN, se lasciato in esecuzione.

Ciò significa che per creare un nuovo database basta fare una chiamata REST **POST** da linea di comando all'indirizzo IP dello stesso server per inserire un documento: dato che il suddetto database non esiste, Elasticsearch provvede a crearlo automaticamente, quindi indicizza il documento ricevuto. Chiamate successive nello stesso formato aggiungeranno nuovi dati all'indice già esistente.

4.2.1 Alcuni comandi REST

Seguono alcuni comandi di esempio [36] [37] che, inviati da una linea di comando Linux, permettono di manipolare i dati del database appena creato con Elasticsearch. Nel client la dicitura `localhost` andrà sostituita con l'indirizzo IP del server, nella rete locale.

- `curl -POST http://localhost:9200/index/doc_type -curl -H 'Content-Type: application/json' -d '{"field1": "value1", "field2": "value2"}'`

Crea un nuovo indice chiamato `index` ed un nuovo tipo di documento, chiamato `doc_type`, ed inserisce un documento formato da due campi con i rispettivi valori (formato JSON). L'ID di tale documento verrà

generato automaticamente da Elasticsearch, in modo che sia unico e possa sempre identificare univocamente i dati appena inseriti.

- `curl -POST http://localhost:9200/index/doc_type/12345 -`
`curl -H 'Content-Type: application/json' -d '{"field1"`
`: "value1", "field2": "value2"}'`

Uguale al comando precedente, tranne per il fatto che in questo caso l'ID non è più generato automaticamente, ma viene specificato dall'utente (e vale 12345).

- `curl -GET http://localhost:9200/index/doc_type/12345`

Ricerca e ritorna il documento con ID pari a 12345.

- `curl -PUT http://localhost:9200/index/doc_type/12345 -`
`curl -H 'Content-Type: application/json' -d '{"field3"`
`: "value3", "field4": "value4"}'`

Modifica (aggiorna) il documento inserito precedentemente; un futuro comando GET mostrerà il numero della nuova versione.

- `curl -POST http://localhost:9200/index/_count`

Conta il numero di documenti presenti nel database.

- `curl -DELETE http://localhost:9200/index`

Cancella l'indice specificato assieme a tutti i suoi documenti.

4.3 Aggiornamento del database

Una volta configurato e avviato il server come descritto al paragrafo 4.1, per popolarlo automaticamente con i dati ricevuti dal SensorTag e salvati

dallo script `connect.py` basta avviare **`process_data.py`** [25], che provvede ad aggiungere al database gli ultimi (eventuali) log non ancora elaborati e ad aggiornare poi continuamente i nuovi dati acquisiti dai sensori, in quattro fasi:

- acquisizione delle impostazioni;
- connessione al database;
- inserimento log vecchi;
- loop di aggiornamento del database.

4.3.1 Acquisizione delle impostazioni

Come per lo script `connect.py`, anche `process_data.py` utilizza un file di impostazioni nella cartella `config`, chiamato **`setup_db.ini`** e contenente tutte le variabili necessarie alla connessione e all'aggiornamento del database. I suoi parametri hanno il seguente significato:

`host`: l'indirizzo IP del server

`port`: la porta del server; solitamente non è necessario modificarla: il default è 9200

`index`: il nome dell'indice in cui saranno inseriti i documenti (ovvero le letture dei sensori)

`doc_type`: il tipo dei documenti che verranno inseriti nell'indice

`notify_index`: il nome dell'indice utilizzato per il logging di connessione/disconnessione dei dispositivi

`notify_doc_type`: il tipo dei documenti che verranno inseriti nell'indice di logging della connessione

`period`: periodo di controllo del log e aggiornamento del database, in millisecondi; ovviamente deve essere uguale o maggiore alla frequenza di lettura dei sensori

`attempts`: numero di tentativi da fare per recuperare il log del giorno corrente, prima di uscire dal programma

`logs_path`: nome della cartella in cui si trovano i log

4.3.2 Connessione al database

La connessione al database avviene tramite la libreria Python **elasticsearch-py** [38] [39], che fornisce i metodi necessari per contattare efficacemente un server Elasticsearch.

Per connettersi quindi basta creare un nuovo oggetto, per semplicità chiamato `es`, fornendo host e porta - in realtà già acquisiti nella fase precedente dal file di impostazioni. Questo elemento potrà essere utilizzato in seguito per eseguire altre operazioni sul database.

4.3.3 Inserimento log vecchi

Il file **last.date**, presente nella cartella `config`, contiene la data e l'ora dell'ultimo aggiornamento del database per ogni dispositivo che abbia contribuito allo stesso database almeno una volta.

Servendosi di questa serie di date e della funzione ausiliaria **modules/min_date.py**, lo script determina qual è stato il giorno meno recente in cui

ha aggiornato il database (a prescindere dal dispositivo), e procede a leggere il log di quel giorno e di tutti quelli successivi (incrementando la data con la funzione `modules/increase_day.py`).

Sebbene possa essere occasionalmente superfluo, il controllo dell'ultimo giorno è necessario per garantire che tutti i dati acquisiti in quel giorno siano stati inviati al database. Potrebbe accadere infatti che lo script si interrompa ad esempio a metà giornata, ed i dati salvati dopo tale interruzione non vengano più inviati al database; di conseguenza essi andrebbero ignorati (e probabilmente perduti) se il log dell'ultimo giorno non venisse riletto integralmente all'avvio successivo di `process_data.py`.

Il programma prosegue quindi con l'inserimento dei log vecchi, controllando che i file effettivamente esistano (dato che i loro nomi vengono generati automaticamente a partire dalla data dell'ultimo aggiornamento, e non è garantito che vengano acquisiti dati tutti i giorni) e, in tal caso, invocando la funzione ausiliaria `modules/update_db.py` per ogni file di dati.

update_db.py è una funzione che prende in ingresso un log, la data e l'ora dell'ultimo aggiornamento, l'oggetto Elasticsearch `es` ed i nomi dell'indice del database e del tipo dei documenti da inserire. Servendosi di questi parametri, lo script invoca a sua volta sul log la funzione **modules/df2json.py** che, come implica il nome, utilizza `modules/raw2df.py` per creare un dataframe Pandas dai dati grezzi e convertirlo poi in un dizionario JSON da ritornare a `update_db.py`.

Tale dizionario rappresenta i dati del log elaborati secondo gli algoritmi presenti nella cartella `sensors`, e riscritti in un formato accettato da Elasticsearch (il JSON, appunto). `update_db.py` quindi prende ogni lettura dei sensori dal dizionario e, dopo aver controllato che la data e l'ora dell'acquisizione del dato considerato siano più recenti della data ed ora dell'ultimo aggiorna-

mento ricevuta come parametro in ingresso, procede con l'inserimento nel database tramite la funzione della libreria Python di Elasticsearch `index(index, doc_type)` invocata sull'oggetto `es` con i corrispondenti parametri (anch'essi ricevuti in ingresso da `process_data.py`).

Per concludere, `update_db.py` modifica la data dell'ultimo aggiornamento nel file `config/last.date`, e ritorna un flag che comunica se ha effettivamente inserito dei dati nel database oppure non ne ha trovati di nuovi.

4.3.4 Loop di aggiornamento

Dopo aver quindi inserito eventuali log vecchi, lo script `process_data.py` entra in un **loop infinito di aggiornamento** del database, leggendo periodicamente (con periodo `period` specificato in `config/setup_db.ini`) il log con la data del giorno corrente e comunicando a video se ad ogni controllo ha inserito dei dati oppure non ne ha trovati di nuovi.

Ad ogni lettura del log lo script controlla che la data corrente sia rimasta la stessa; se dovesse rilevare un cambiamento, procede allora ad incrementarla e leggere il log del nuovo giorno.

Nota: Nel caso in cui non esistesse il log del giorno corrente (perché il dispositivo non è ancora stato avviato oppure durante la giornata precedente ha smesso di funzionare e quindi non ha creato un nuovo log dopo la mezzanotte), il programma continua a cercarlo per `attempts` volte (variabile in `setup_db.ini`, vedi 4.3.1) e, in caso di insuccesso, termina automaticamente.

Capitolo 5

Visualizzazione dei dati

La scelta di Elasticsearch come server e database per il presente sistema IoT si è rivelata conveniente anche durante la fase finale del progetto, in quanto Elastic fornisce, come parte di un insieme di applicazioni per la gestione dei server chiamato *The ELK Stack* (acronimo di Elasticsearch, Logstash e Kibana [40]), anche uno strumento per la visualizzazione grafica dei dati: **Kibana**.



Figura 5.1: In inglese *elk* significa anche *alce*.

Kibana [41] è integrato con Elasticsearch al punto che richiede uno sforzo

minimo per essere configurato, e permette di creare viste avanzate dei dati presenti in un qualsiasi index del server su cui è installato. Si tratta di un tool molto potente che ha tutti i requisiti per essere utilizzato in ambito professionale, anche per enormi quantità di dati.

5.1 Installazione e configurazione di Kibana

Per installare Kibana, dunque, è sufficiente scaricarlo dal sito `elastic.co` [41] ed estrarlo in una qualunque posizione sul disco fisso (possibilmente insieme a Elasticsearch). Per configurarlo basta aprire il file `/kibana/config/kibana.yml` per impostare l'indirizzo del server, come segue (il default è `localhost`, per un server installato nella stessa macchina di Kibana):

```
elasticsearch.hosts: ["http://localhost:9200"]
```

Come per la configurazione di Elasticsearch, bisogna **cancellare il commento** `#` presente di default davanti alla nuova impostazione, che altrimenti non avrà alcun effetto.

5.1.1 Avvio e accesso

Per avviare Kibana bisogna aprire il file `bin/kibana` (oppure `bin/kibana.bat` su Windows), e **lasciarlo in esecuzione**.

A questo punto Kibana è accessibile dal browser all'indirizzo **`http://localhost:5601`**; da qui è possibile utilizzare tutte le funzioni disponibili direttamente dall'interfaccia, di cui alcune vengono dettagliate di seguito.

```

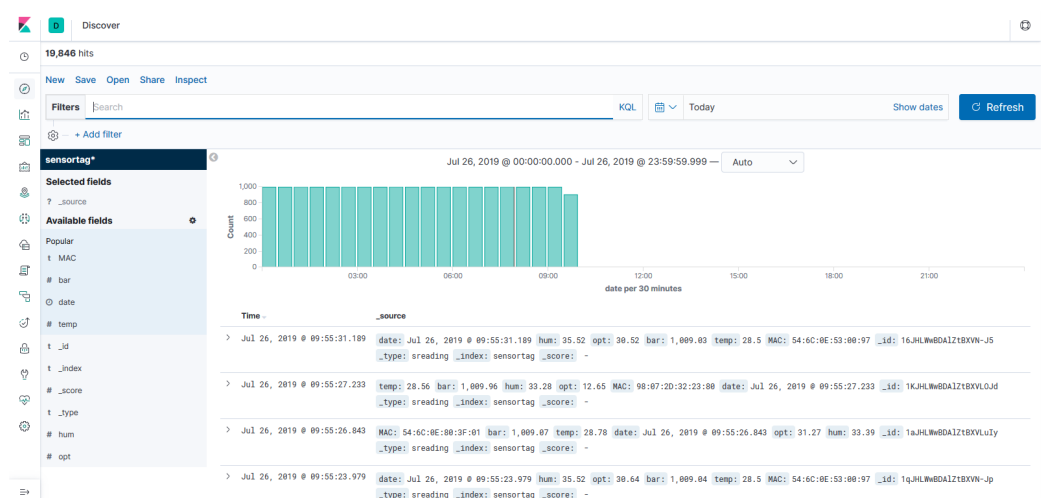
log [13:48:06.167] [info][status][plugin:watcher@7.2.0] Status changed from red to green - Ready
log [13:48:06.168] [info][status][plugin:grokdebugger@7.2.0] Status changed from red to green - Ready
log [13:48:06.169] [info][status][plugin:logstash@7.2.0] Status changed from red to green - Ready
log [13:48:06.171] [info][status][plugin:beats_management@7.2.0] Status changed from red to green - Ready
log [13:48:06.172] [info][status][plugin:index_management@7.2.0] Status changed from red to green - Ready
log [13:48:06.174] [info][status][plugin:index_lifecycle_management@7.2.0] Status changed from red to green - Ready
log [13:48:06.175] [info][status][plugin:rollup@7.2.0] Status changed from red to green - Ready
log [13:48:06.178] [info][status][plugin:remote_clusters@7.2.0] Status changed from red to green - Ready
log [13:48:06.179] [info][status][plugin:cross_cluster_replication@7.2.0] Status changed from red to green - Ready
log [13:48:06.180] [info][status][plugin:snapshot_restore@7.2.0] Status changed from red to green - Ready
log [13:48:06.182] [info][status][plugin:reporting@7.2.0] Status changed from red to green - Ready
log [13:48:06.183] [info][kibana-monitoring][monitoring] Starting monitoring stats collection
log [13:48:06.187] [info][status][plugin:maps@7.2.0] Status changed from red to green - Ready
log [13:48:06.188] [info][status][plugin:security@7.2.0] Status changed from red to green - Ready
log [13:48:06.282] [error][status][plugin:spaces@7.2.0] Status changed from red to red - No shard available for [get
[kibana][_doc][space:default]: routing [null]]: [no_shard_available_action_exception] No shard available for [get [.ki
bana][_doc][space:default]: routing [null]]
log [13:48:06.454] [error][task_manager] Failed to poll for work: [search_phase_execution_exception] all shards fail
ed :: {"path":"/.kibana_task_manager/_search","query":{"ignore_unavailable":true},"body":{"query":{"bool":{"must":
:[{"term":{"type":"task"}},{\bool":{"must":[{"terms":{"task.taskType":["maps telemetry","vis telemetry"
]}},{\range":{"task.attempts":{"lte":3}}},{\range":{"task.runAt":{"lte":"now"}}},{\range":{"kibana.api
Version":{"lte":1}}}]}}]},"size":10,"sort":{"task.runAt":{"order":"asc"},"seq_no_primary_term":{"true"},"
statusCode":503,"response":{"error":{"root_cause":[{"type":"search_phase_execution_exception","reason":{"all
shards failed"},"phase":"query","grouped":true,"failed_shards":[]},"status":503}}}}
log [13:48:07.093] [error][status][plugin:spaces@7.2.0] Status changed from red to red - No shard available for [get
[kibana][_doc][space:default]: routing [null]]: [illegal_index_shard_state_exception] CurrentState[RECOVERING] operati
ons only allowed when shard state is one of [POST_RECOVERY, STARTED], with { index_uuid="gw1LAXEWRZuAX_W2JfLL4w" & shard
="0" & index=".kibana_1" }
log [13:48:07.526] [info][status][plugin:spaces@7.2.0] Status changed from red to green - Ready

```

Figura 5.2: Processo `kibana.bat` a server Elasticsearch in funzione.

5.2 Visualizzazioni

La scheda *Discover* di Kibana mostra, in formato grezzo, gli ultimi dati aggiunti al database (fig. 5.3). La vista può essere modificata a piacere cliccando sull'icona del calendario e poi su *Refresh* per visualizzare i dati inseriti in un certo periodo di tempo.

Figura 5.3: Scheda *Discover* di Kibana.

Nuove **visualizzazioni** (*visualizations*) possono essere create nella scheda *Visualize*. Cliccando su *Create new visualization* è infatti possibile selezionare un tipo di grafico (nella figura qui sotto è stato usato il tipo *Line*), scegliere l'indice da cui estrarre i dati ed in seguito impostare assi e relative metriche.

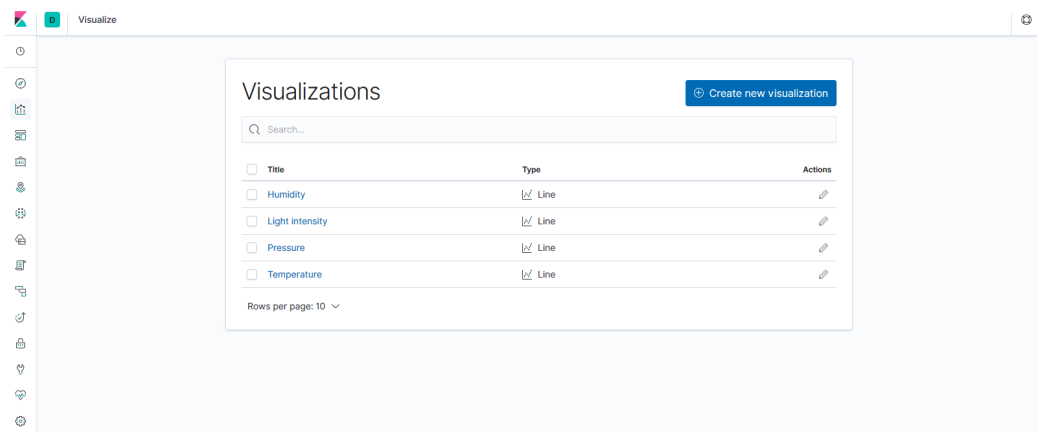


Figura 5.4: Scheda *Visualizations* di Kibana.



Figura 5.5: Una visualizzazione della temperatura ambientale.

Per ottenere la visualizzazione della figura 5.5 sono state utilizzate le seguenti impostazioni:

Data

Metrics (Y-Axis)

Aggregation: Average

Field: temp

Custom label: Ambient temperature (°C)

Buckets (X-Axis)

Aggregation: Date Histogram

Field: date

Minimum Interval: Auto

Metrics & Axes

Chart Type: line

Mode: normal

Value Axis: LeftAxis-1

Line Mode: smoothed

Show Line: yes

Show Circles: yes

Line Width: 2

È possibile suddividere i dati del grafico a linea anche in base ad altri campi, come ad esempio l'indirizzo MAC dei dispositivi (fig. 5.6). Per farlo bisogna cliccare il pulsante *Add sub-buckets* nella sezione *Data* e selezionare *Split Series* con le seguenti impostazioni:

Data

Sub aggregation: Terms

Field: MAC.keyword

Order by: metric: Ambient temperature (°C)

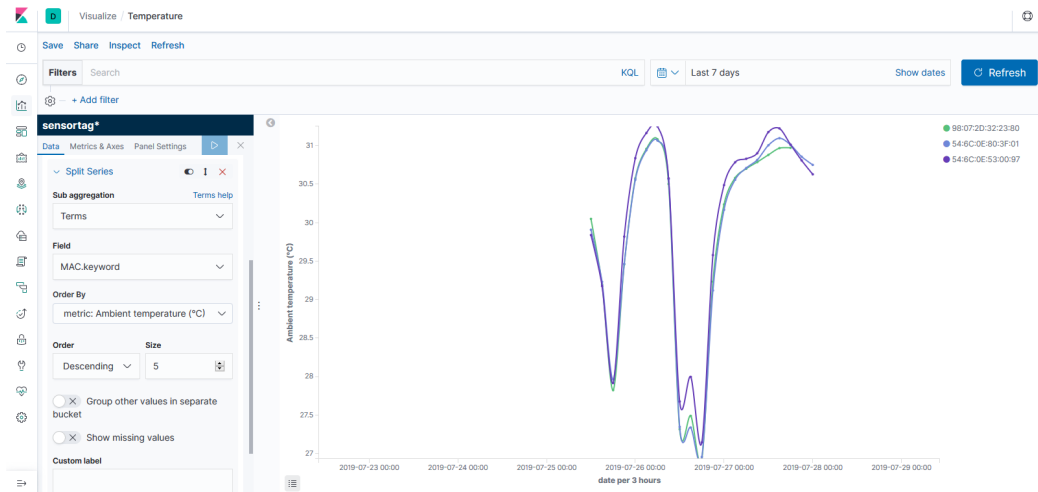
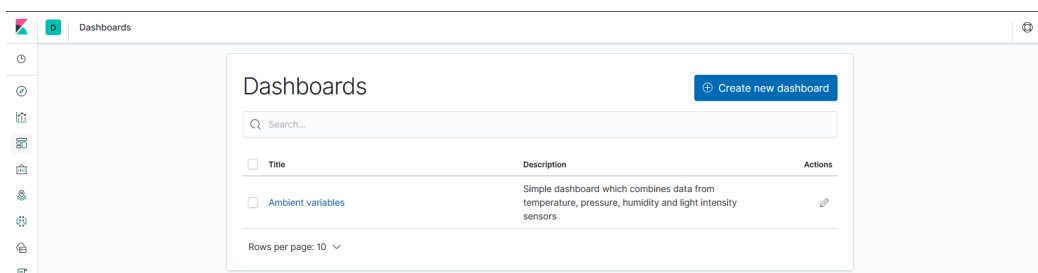


Figura 5.6: Temperatura ambientale suddivisa per ogni SensorTag.

Ulteriori opzioni per le visualizzazioni sono descritte nell'apposita pagina della documentazione ufficiale di Kibana [42]. È importante, prima di uscire, ricordarsi di **salvare** la visualizzazione appena creata (*Save*, in alto a sinistra).

5.3 Dashboard

La dashboard è uno strumento utile per creare gruppi di visualizzazioni in relazione tra loro ed averle così sempre pronte per la consultazione.

Figura 5.7: Scheda *Dashboards* di Kibana.

Per crearne una bisogna cliccare, nella scheda *Dashboard*, su *Create new dashboard* e, nella pagina che si apre in seguito, su *Add* in alto a sinistra.

A questo punto basterà selezionare una o più tra le visualizzazioni presenti nell'elenco (se quest'ultimo fosse vuoto significa che non ci sono visualizzazioni, e che ne andranno create alcune nel modo descritto precedentemente); Kibana creerà la nuova dashboard, che permetterà di vedere a colpo d'occhio più informazioni mappate sulla stessa data (selezionabile, come nella scheda *Discovery*, dall'icona del calendario in alto). Ovviamente, prima di uscire, bisogna salvare la nuova dashboard cliccando su *Save* (in alto a sinistra).

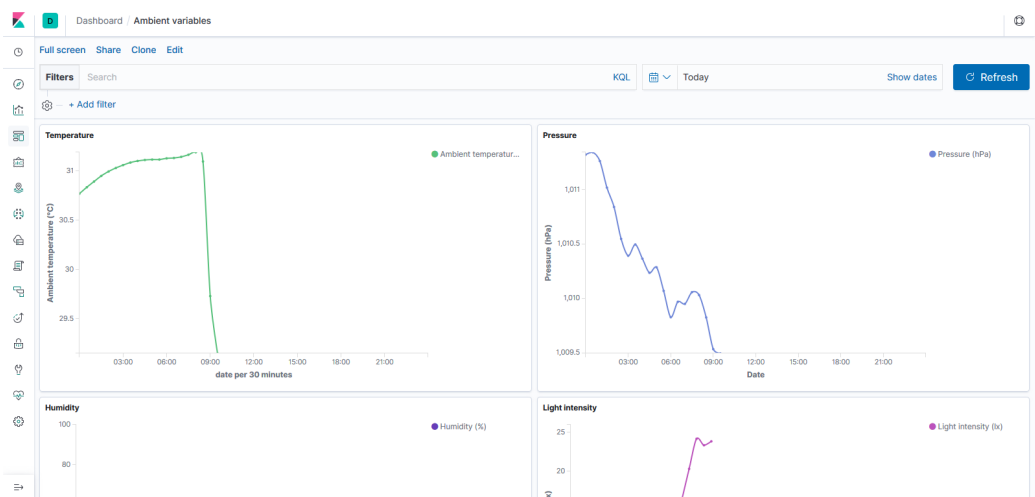


Figura 5.8: Una dashboard contenente le variabili d'ambiente rilevate dai SensorTag.

Ulteriori dettagli sulla gestione delle dashboard sono presenti nella guida [42] di Kibana.

5.4 Indici

Il database per l'archiviazione dei dati acquisiti dai SensorTag, già elaborati, è stato creato e mantenuto grazie allo script Python `process_data.py`, come descritto nel capitolo 4. Per farlo è stato utilizzato un unico indice Elasticsearch, chiamato semplicemente **sensortag**, ed il tipo dei documenti inse-

riti (ovvero le singole letture dei sensori) è stato arbitrariamente denominato **sreading** (abbreviazione per *sensor reading*).

Al fine di monitorare il corretto funzionamento del sistema e di individuare eventuali malfunzionamenti, nell'ottica di estendere l'implementazione di alcuni accorgimenti nella categoria del **Quality of Service** (QoS) - già costituito da diversi meccanismi di riconnessione e controllo errori, vedi par. 3.2.5 - è stato aggiunto un secondo indice chiamato **connection_log** (fig. 5.9). Tale database conserva la data e l'ora di ogni connessione al Raspberry, così come di ogni disconnessione, specificando per ognuna in parte l'indirizzo MAC del dispositivo e l'esito dell'operazione.

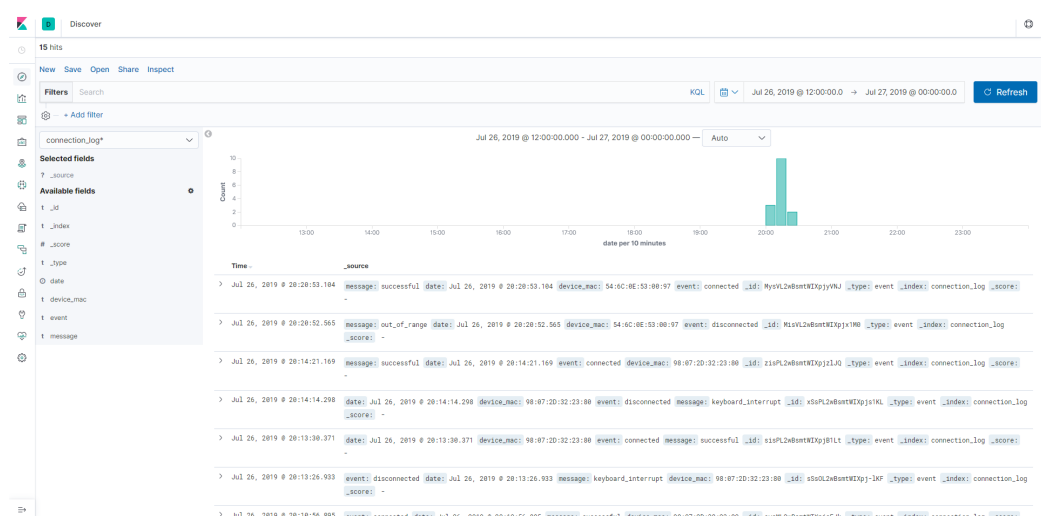


Figura 5.9: La scheda *Discover* dell'indice `connection_log`.

Capitolo 6

Conclusioni

In questo capitolo vengono discussi i difetti dei SensorTag utilizzati per il progetto, alcune linee guida sulla scelta di un dispositivo adatto ad essere installato in cantiere ed un paio di possibili miglioramenti del sistema creato nell'ambito della tesi in oggetto.

6.1 Range e batteria del SensorTag

Una serie di test empirici ha mostrato che il range BLE del SensorTag non è molto ampio. Sebbene il segnale risulti sufficiente in qualunque punto di una stanza da circa 20 m² (anche dentro mobili o cassetti), allontanarsi di pochi altri metri (3-4 m) dal Raspberry significa rischiare di perdere la connessione. Ciò accade sicuramente non appena si frappone un muro tra SensorTag e collettore, anche non molto spesso (come ad esempio un muro non portante).

Ne consegue che per ottimizzare la connessione è consigliabile non porre il SensorTag troppo lontano dal Raspberry, e soprattutto assicurarsi che tra i due dispositivi, in linea d'aria, non ci siano oggetti che possano bloccare il

segnale. Monitorare il corretto funzionamento del SensorTag per alcune ore, controllando la presenza di eventuali errori nel database delle connessioni (indice `connection_log` di Elasticsearch), può essere un valido metodo per determinare se la posizione scelta è adatta oppure tende a causare disconnessioni occasionali.

Similmente, nemmeno la batteria del SensorTag si presta ad utilizzi prolungati. Nonostante le specifiche BLE vantino un basso consumo di energia, [7] [43] è stato osservato durante i test e la raccolta dei dati per il popolamento del database che le piccole batterie CR2032 che alimentano i SensorTag durano all'incirca 55 ore, ovvero poco più di due giorni. Questo risultato è stato ottenuto con un intervallo di polling dei dati di 5 secondi; si stima che l'aumento del periodo di richiesta dei dati consenta una durata della batteria maggiorata in proporzione allo stesso.

In ogni caso, è evidente come il SensorTag non sia adatto ad essere utilizzato per raccogliere dati in tempo quasi reale, sia per il range Bluetooth limitato che per la scarsa durata della batteria, fattori che insieme comportano un'inaffidabilità non accettabile in applicazioni industriali. Vengono quindi esplorate in seguito alcune opzioni alternative a questo dispositivo.

6.2 Soluzioni concrete

Il mondo dell'IoT è in costante evoluzione: il basso costo della maggior parte dei dispositivi disponibili sul mercato e la crescente domanda di soluzioni più avanzate fanno sì che sempre più aziende sviluppino il proprio hardware e software per creare sistemi personalizzati [44]. La conseguenza immediata di questa tendenza è stata l'ideazione di innumerevoli dispositivi dalle caratteristiche più disparate, spesso simili ma comunque abbastanza

differenti affinché non siano sempre compatibili tra loro.

Proprio a causa di questa enorme varietà è assai difficile scegliere l'hardware giusto per creare un sistema IoT, in quanto bisogna avere un'ottima conoscenza del caso d'uso e quindi dei requisiti che i dispositivi dovranno rispettare. Spesso si rende necessaria un'analisi preliminare approfondita dell'ambiente in cui il sistema verrà installato, al fine di individuare eventuali problemi e criticità, così come le loro possibili soluzioni.

Come già specificato estensivamente in precedenza, il progetto sviluppato per la presente tesi è soltanto un prototipo di quello che sarà un sistema complesso da installare nei cantieri edili, nell'ambito delle direttive europee e regionali per lo sviluppo e l'integrazione dell'Industria 4.0. Avendo per questo motivo utilizzato quasi esclusivamente sensori adatti al solo uso domestico, si rende quindi necessario individuare alcuni possibili casi d'uso resi possibili dai dispositivi attualmente presenti sul mercato, che possano effettivamente vedere un'applicazione concreta in cantiere.

Innanzitutto, il requisito fondamentale per qualunque dispositivo che debba essere impiegato in questo scenario è il fatto che deve essere in grado di operare anche in condizioni ambientali avverse, ovvero a temperature particolarmente basse oppure alte ed in presenza di pioggia, polvere e vibrazioni. Di conseguenza la scelta dei sensori dovrà escludere tutti quei apparecchi progettati per un utilizzo domestico che rischiano di andare incontro a malfunzionamenti se sottoposti a tali fattori, oppure dovrà prevedere la loro protezione attraverso adeguate misure di schematura.

È possibile infatti impiegare anche dispositivi non esplicitamente adatti ad un utilizzo esterno attraverso la loro collocazione in apposite scatole di derivazione, in modo che siano riparati dagli agenti meteorologici. L'applicazione più comune di questa soluzione riguarda soprattutto gli accentratori dati,

come il Raspberry Pi utilizzato nel presente progetto, che possono così essere disposti sul campo rimanendo comunque protetti entro ragionevoli limiti da possibili infiltrazioni di acqua e polvere, così come dai raggi solari e dalle temperature estreme. Ovviamente un sistema del genere richiederà una manutenzione periodica, al fine di verificare l'effettivo isolamento del dispositivo e di effettuare una pulizia che, inevitabilmente, si renderà necessaria.

Nel caso di sensori che misurano direttamente le variabili d'ambiente, come ad esempio la temperatura oppure le quantità di gas nell'aria, il posizionamento del dispositivo diventa cruciale per il suo corretto funzionamento. Alcuni di questi sensori infatti necessitano di essere a diretto contatto con l'atmosfera circostante, perciò isolarli in una scatola di derivazione potrebbe portare a rilevamenti errati che non riflettono la situazione reale. Nel caso della temperatura è sufficiente una sonda protetta da pasta termoconduttiva, mentre in altri casi si rendono necessari dispositivi con una certa affidabilità, che inevitabilmente comportano un costo maggiore rispetto a quelli comunemente reperibili in commercio e che anzi, risultano spesso introvabili e/o inadatti al proprio caso d'uso; ne consegue che un sistema IoT di livello industriale solitamente richiede l'intervento di qualche azienda IT in grado di progettare soluzioni personalizzate.

Il prototipo discusso nella presente tesi rientra proprio in quest'ultimo caso: il suo scopo, infatti, è quello di esplorare il funzionamento dei sistemi IoT in contesti di laboratorio, per spostarli gradualmente in ambienti industriali ed infine proporli a potenziali clienti alla ricerca di una soluzione professionale progettata su misura.

6.3 Estensione del progetto

Durante lo sviluppo del presente progetto sono state individuate alcune criticità che, sebbene siano state risolte in corso d'opera, hanno portato a riflessioni ed un paio di spunti per alcuni possibili miglioramenti futuri.

Il primo tra questi è sicuramente la scelta del dispositivo di rilevazione dati, il quale si è rivelato assai inadatto soprattutto per la bassa connettività e durata della batteria. Una soluzione al consumo di energia potrebbe essere l'adozione di un dispositivo alimentato ad energia solare e quindi autonomo, così come una migliore connettività sarebbe offerta da protocolli diversi dal Bluetooth LE, come ad esempio Zigbee, LPWAN (Low Power Wide Area), Wi-Fi oppure reti cellulari (4G/5G).

L'altro possibile miglioramento consisterebbe nell'utilizzo, per il logging della connettività, dell'applicazione Elastic Logstash [45]. Questo comporterebbe una completa integrazione del progetto con lo stack ELK [40], così come la disponibilità di uno strumento potente di logging in grado di andare oltre la semplice visualizzazione (a differenza della soluzione attualmente implementata), ovvero capace di analizzare accuratamente gli eventi di connessione dei SensorTag e fornire una loro ampia analisi.

6.4 Considerazioni finali

Il progetto presentato in questo lavoro ha costituito un punto di partenza fondamentale per lo studio dei sistemi IoT e, più in generale, del funzionamento e utilizzo di diversi strumenti e tecnologie tra cui il protocollo Bluetooth LE, gatttool, Pexpect, lo stack ELK, e, non ultimo, il sistema operativo Linux. Le conoscenze acquisite durante il suo sviluppo hanno permesso

di evidenziare pregi e difetti dei dispositivi usati, così come di analizzare i risultati ottenuti per poter delineare alcuni possibili sviluppi futuri.

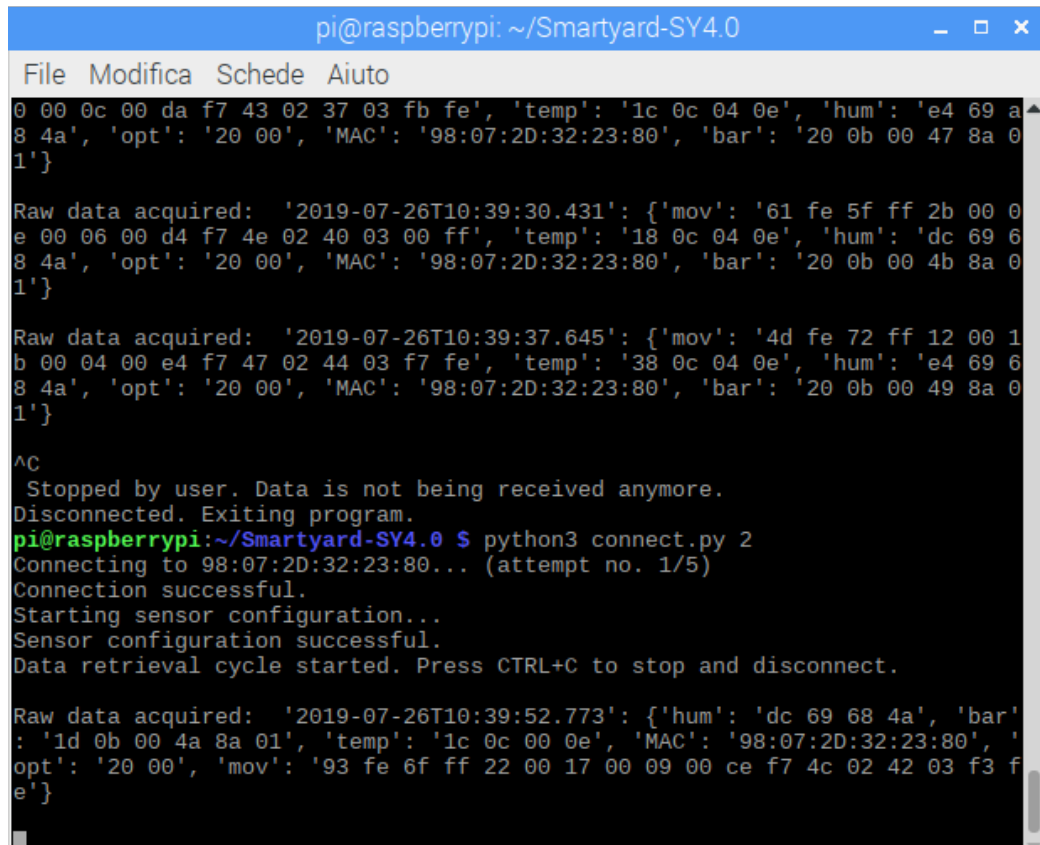
Il software, di per sé, è stato progettato con l'intento di essere molto flessibile, in modo che sia compatibile anche con dispositivi diversi dal SensorTag (con la connettività BLE come unico requisito). L'obiettivo finale, ovvero quello di creare un sistema IoT completo, dal livello più basso costituito dai sensori, passando attraverso l'accentratore per arrivare ad un server web in grado di archiviare e visualizzare i dati, è stato raggiunto con successo. Il codice sorgente nella repository GitHub [25] è a disposizione di chiunque voglia ricreare il sistema, con la presente tesi utilizzabile come documentazione e guida all'installazione.

I dati archiviati durante i vari test sono stati usati per esplorare le funzionalità di Kibana; analizzando i risultati ottenuti anche attraverso confronti diretti con le condizioni ambientali (come ad esempio la temperatura segnata dal termostato, oppure la luminosità media in un ufficio [46]) si verifica facilmente il corretto funzionamento del sistema nel suo complesso.

Riscrivendo quindi adeguatamente i parametri dei file di configurazione (`setup.ini` e `setup_db.ini`, vedi cap. 3.2.1), i programmi sviluppati sono utilizzabili con qualunque dispositivo BLE e su qualunque macchina Linux come accentratore e/o server. Ulteriormente, i file sorgente possono essere modificati ad hoc per ampliare le funzionalità oppure toglierne alcune (come ad esempio la connessione ad un server Elasticsearch); l'estensiva documentazione del codice attraverso commenti interni è stata appositamente pensata per facilitare queste modifiche ai programmatori.

In conclusione, lo sviluppo di questo prototipo non ha avuto soltanto un importante valore didattico, ma ha portato anche al completamento di uno strumento di utilità generale; seguono quindi alcune immagini che mostrano

il programma in esecuzione, per illustrare concretamente il funzionamento del sistema.



```
pi@raspberrypi: ~/Smartyard-SY4.0
File Modifica Schede Aiuto
0 00 0c 00 da f7 43 02 37 03 fb fe', 'temp': '1c 0c 04 0e', 'hum': 'e4 69 a
8 4a', 'opt': '20 00', 'MAC': '98:07:2D:32:23:80', 'bar': '20 0b 00 47 8a 0
1'}

Raw data acquired: '2019-07-26T10:39:30.431': {'mov': '61 fe 5f ff 2b 00 0
e 00 06 00 d4 f7 4e 02 40 03 00 ff', 'temp': '18 0c 04 0e', 'hum': 'dc 69 6
8 4a', 'opt': '20 00', 'MAC': '98:07:2D:32:23:80', 'bar': '20 0b 00 4b 8a 0
1'}

Raw data acquired: '2019-07-26T10:39:37.645': {'mov': '4d fe 72 ff 12 00 1
b 00 04 00 e4 f7 47 02 44 03 f7 fe', 'temp': '38 0c 04 0e', 'hum': 'e4 69 6
8 4a', 'opt': '20 00', 'MAC': '98:07:2D:32:23:80', 'bar': '20 0b 00 49 8a 0
1'}

^C
Stopped by user. Data is not being received anymore.
Disconnected. Exiting program.
pi@raspberrypi:~/Smartyard-SY4.0 $ python3 connect.py 2
Connecting to 98:07:2D:32:23:80... (attempt no. 1/5)
Connection successful.
Starting sensor configuration...
Sensor configuration successful.
Data retrieval cycle started. Press CTRL+C to stop and disconnect.

Raw data acquired: '2019-07-26T10:39:52.773': {'hum': 'dc 69 68 4a', 'bar'
: '1d 0b 00 4a 8a 01', 'temp': '1c 0c 00 0e', 'MAC': '98:07:2D:32:23:80', '
opt': '20 00', 'mov': '93 fe 6f ff 22 00 17 00 09 00 ce f7 4c 02 42 03 f3 f
e'}
```

Figura 6.1: **connect.py**. Disconnessione causata dall'utente durante il ciclo di acquisizione dati e riconnessione manuale.

```

pi@raspberrypi: ~/Smartyard-SY4.0
File Modifica Schede Aiuto
Checking for new data...
Data has been successfully added to database. (2019-07-26 10:39:05.156)
Waiting for next check...

Checking for new data...
Data has been successfully added to database. (2019-07-26 10:39:18.156)
Waiting for next check...

^C
Stopped by user. Database is not being updated anymore. Exiting program.
pi@raspberrypi:~/Smartyard-SY4.0 $ python3 process_data.py 3
Connecting to Elasticsearch database...
Connection successful.

Checking for old logs...
Old logs check successful.
Old logs successfully added to database.

Starting data processing loop.

Checking for new data...
Data has been successfully added to database. (2019-07-26 10:39:33.156)
Waiting for next check...

```

Figura 6.2: `process_data.py`. Interruzione del processo di elaborazione e archiviazione dati e successivo riavvio manuale.

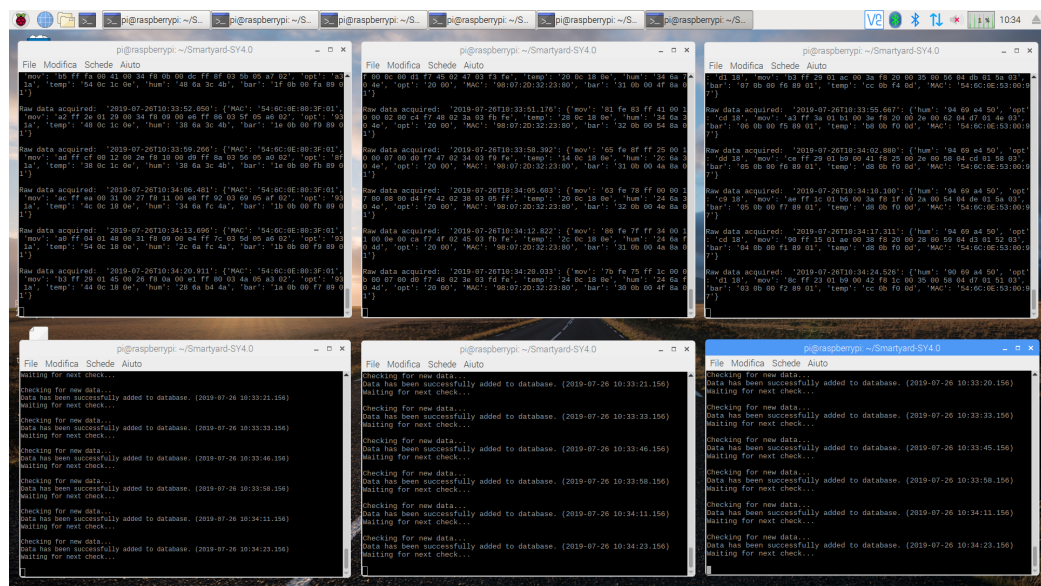


Figura 6.3: Connessione di **tre SensorTag** contemporaneamente e altrettanti processi di elaborazione dei dati in funzione sul Raspberry Pi.

Bibliografia

- [1] Mark Hung, “Leading the IoT,” *Gartner Inc.*, 2017.
- [2] FORMEDIL/CNCPT, “Rapporto Attività 2018 FORMEDIL/CNCPT,” Nov 2018.
- [3] “Magenta s.r.l.” <https://www.magentalab.it/>.
- [4] “Smartyard - SY 4.0.” <https://www.sy-quattropuntozero.it/>.
- [5] Magenta s.r.l., “SY 4.0 - Smartyard.” https://www.magentalab.it/?page_id=832.
- [6] Regione Toscana, “Indirizzi per l’attuazione della strategia industria 4.0.,” Nov 2016.
- [7] Bluetooth SIG, “Radio Versions,” *Bluetooth.com*.
- [8] Bluetooth SIG, “Our History,” *Bluetooth.com*.
- [9] Bluetooth SIG, “Core Specifications,” *Bluetooth.com*.
- [10] Elvis Pfützenreuter, “Bluetooth: ATT and GATT,” *epxx.co*.
- [11] Bluetooth SIG, “GATT Specifications,” *Bluetooth.com*.
- [12] Bluetooth SIG, “GATT Characteristics,” *Bluetooth.com*.

-
- [13] Bluetooth SIG, “GATT Services,” *Bluetooth.com*.
 - [14] “Texas Instruments SimpleLink SensorTag CC2650STK.” www.ti.com/tool/CC2650STK.
 - [15] Texas Instruments, “SimpleLink SensorTag CC2650STK Data Sheet,”
 - [16] Michael Setton, Jarle Boe, “A Guide to SensorTag Hackathons: Resources,” *Texas Instruments*, 2016.
 - [17] “Raspberry Pi 3 Model B+.” <https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/>.
 - [18] Raspberry Pi Foundation, “Raspberry Pi 3 Model B+ Data Sheet,”
 - [19] Tony DiCola, “Install bluez on the Raspberry Pi,” *Adafruit*.
 - [20] Allan Marube, “TI Sensor Tag and Raspberry Pi,” *IBM*.
 - [21] Nobuhiro Iwamatsu, “gatttool manual,” *Ubuntu Manuals*.
 - [22] Texas Instruments, “CC2650 SensorTag User’s Guide,” *Texas Instruments Wiki*.
 - [23] “Python.” <https://www.python.org/>.
 - [24] Noah Spurrier and contributors, “Pexpect 4.7,” *Read the Docs*.
 - [25] Paula Mihalcea, “Smartyard-SY4.0,” *GitHub*.
 - [26] Python Software Foundation, “ConfigParser,” *Python Documentation*.
 - [27] Erich Styger, “Using Python, Gatttool and Bluetooth Low Energy with Hexiwear,” *MCU on Eclipse*.

-
- [28] International Organization for Standardization, “ISO 8601-2:2019,” *ISO.org*.
- [29] “Python Data Analysis Library,” *PyData*.
- [30] “pyplot,” *matplotlib*.
- [31] “Elasticsearch.” <https://www.elastic.co/downloads/elasticsearch>.
- [32] The Apache Software Foundation, “Apache Lucene.” <https://lucene.apache.org/>.
- [33] Roy Thomas Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” *Doctoral dissertation, University of California, 2000*.
- [34] Samuel Scott, “These 15 Tech Companies Chose the ELK Stack Over Proprietary Logging Software,” *logz.io*.
- [35] Daniel Stenberg, “Everything curl,” *Alibris*.
- [36] Phil Vuollet, “Elasticsearch Tutorial: Your Detailed Guide to Getting Started,” *Stackify*.
- [37] Jason Zucchetto, “Elasticsearch: Getting Started,” *Elastic*.
- [38] Honza Král, “Python Elasticsearch Client,” *Read the Docs*.
- [39] Ernesto Rodríguez, “Python Elasticsearch Client,” *Tryolabs*.
- [40] Elastic, “What is the ELK Stack?,” *Elastic*.
- [41] “Kibana.” <https://www.elastic.co/products/kibana>.

-
- [42] Elastic, “Kibana User Guide,” *Elastic*.
- [43] Nick Koudas, “The Hitchhikers Guide to iBeacon Hardware: A Comprehensive Report by Aislelabs (2015),” *Aislelabs*.
- [44] Louis Columbus, “Top 25 IoT Startups To Watch In 2019,” *Forbes*.
- [45] “Logstash.” <https://www.elastic.co/products/logstash>.
- [46] “Lux.” <https://en.wikipedia.org/wiki/Lux#Illuminance>.

Ringraziamenti

Il presente lavoro conclude un percorso di studi iniziato ben sedici anni fa, ed è perciò giusto dedicare una piccola sezione della tesi per ringraziare almeno una parte delle persone che ho incrociato durante questo lungo viaggio.

Vorrei innanzitutto ringraziare mia madre, senza la quale probabilmente vivrei ancora da qualche parte in Romania. Grazie per avermi permesso, da sola, di proseguire gli studi senza farmi mancare nulla, e di essermi stata accanto in ogni decisione, anche quelle che inizialmente non approvavi e che il tempo ha mostrato invece come giuste (o sbagliate... succede).

Ringrazio Marco, fidato compagno di viaggio da ormai tre anni, che mi sostiene e sopporta e non manca mai di ascoltarmi quando ne ho bisogno: sei un faro incrollabile nel mare tempestoso della vita.

Ringrazio Andrea e Chiara, compagni di chiacchiere e moto senza i quali non avrei mai scritto il curriculum così terribilmente interessante grazie al quale ho già ottenuto un colloquio.

Ringrazio Timothy, che in sole tre settimane mi ha insegnato a smontare e riparare computer e dispositivi di ogni tipo, stimolando così oltremodo il mio interesse verso l'ingegneria informatica.

Ringrazio Federico, tutor fuori dalle righe che non si scompone alle mie domande ma anzi, spiega al meglio anche i concetti più ostici e noiosi senza

mai volere nulla in cambio.

Ringrazio tutti i miei amici, in particolare il gruppo D&D, per aver avuto la pazienza di rimandare uscite e sessioni quando dovevo studiare per gli esami. Sebbene siate stati gentilissimi, devo darvi una brutta notizia: non ho ancora finito. Tenete duro insieme a me un altro paio d'anni.

Ringrazio tutti i professori che mi hanno accompagnata nel mio percorso scolastico sostenendomi ed aiutandomi a crescere, in particolare le proff.sse Marchetti e Benigni ed i proff. Ceccanti e Rosati: ce ne vorrebbero di più come voi.

Ringrazio il prof. A. D. Bagdanov e la sua infinita pazienza: non ha mai mancato di chiarire i miei dubbi e condividere osservazioni e suggerimenti, ed è sempre stato disponibile e disposto al dialogo nonostante i mille impegni; senza di lui non avrei mai concluso il tirocinio entro agosto.

Ringrazio Magenta s.r.l. e tutto lo staff, in particolare Walter; è soltanto grazie a loro che sono entrata materialmente nel mondo dell'IoT, e senza la loro guida avrei errato (o più propriamente, smattato) molto, molto di più. È stato un piacere lavorare con tutti voi.

Infine, non posso che ricordare tutti quei professori che, a qualche punto della mia vita, mi hanno motivata chiedendomi se ero davvero sicura di volermi iscrivere a ingegneria, mostrando più o meno velatamente dei dubbi riguardo le mie capacità: tiè, laureata e pure in corso!