Agenzia nazionale per le nuove tecnologie, l'energia e lo sviluppo economico sostenibile

MINISTERO DELLA TRANSIZIONE ECOLOGICA

Ricerca di Sistema elettrico

Completamento e test del sistema di programmazione, gestione e monitoraggio sul campo della distribuzione urbana delle merci con veicoli elettrici

G.Tomasino, F.Ortenzi, M.Pollino, V.Rosato, M.Corazza, V. Conti, M.P. Valentini

RdS/PTR(2021)/202

COMPLETAMENTO E TEST DEL SISTEMA DI PROGRAMMAZIONE, GESTIONE E MONITORAGGIO SUL CAMPO DELLA DISTRIBUZIONE URBANA DELLE MERCI CON VEICOLI ELETTRICI

G.Tomasino, F.Ortenzi, M.Pollino, V.Rosato (ENEA)
M.Corazza, V. Conti, M.P. Valentini (ENEA)

Dicembre 2021

Report Ricerca di Sistema Elettrico

Accordo di Programma Ministero dello Sviluppo Economico - ENEA
Piano Triennale di Realizzazione 2019-2021 - III annualità 2021
Obiettivo: Tecnologie
Progetto: Tecnologie per la penetrazione efficiente del vettore elettrico negli usi finali
Work package: Mobilità
Linea di attività: la 2.21 Integrazione delle nuove funzionalità nella piattaforma e testing
Responsabile del Progetto: Claudia Meloni, ENEA
Responsabile del Work package: Maria Pia Valentini, ENEA

Indice

Sommario

In questa terza annualità sono state condotte delle prove sperimentali all'interno dell'area del Centro di Ricerca ENEA della Casaccia in Roma, per verificare sul campo le funzionalità progettate nelle annualità precedenti relativamente al monitoraggio dei veicoli elettrici adibiti alla distribuzione delle merci in città, attraverso impiego di sensori installati sul campo e all'interno dei veicoli.

Allo stesso tempo sono state portate a termine le attività relative alla realizzazione e alla verifica di un sistema per la programmazione e la gestione delle consegne merci per mezzo di una flotta di veicoli elettrici, con particolare attenzione alla necessità di ricarica dei veicoli e alla gestione degli scostamenti dalle condizioni di viaggio previste.

Vengono riportati nel presente documento i risultati conseguiti.

1 Introduzione

Lo scenario di riferimento in cui si collocano gli strumenti messi a punto e collaudati in questa Linea di Attività è quello della Smart Mobility applicata all'interno di un'area urbana. Più in particolare si affronta il tema della distribuzione urbana delle merci operata attraverso flotte di veicoli elettrici.

L'obiettivo è di ottimizzare, giorno per giorno, i giri di distribuzione delle consegne all'interno di una rete urbana, tenendo conto dei vincoli temporali imposti dai destinatari della merce (Time – Windows), della capacità di carico e dell'autonomia dei veicoli, considerando anche la possibilità di ricaricare la batteria attraverso stazioni di ricarica veloce sparse sul territorio.

Inoltre, si vogliono gestire eventuali imprevisti derivanti da anomale condizioni di traffico o di percorribilità della rete stradale che potrebbero richiedere variazioni in tempo reale della programmazione originaria, dando luogo a nuovi percorsi e/o operazioni di ricarica.

In quest'ottica rivestono importanza fondamentale le informazioni che è possibile acquisire in tempo reale sia dal veicolo in movimento che dal territorio nel quale esso opera.

Per quanto riguarda il veicolo, le informazioni di interesse sono tutte quelle inerenti la posizione istantanea e lo stato di carica della batteria. Per quanto riguarda invece il territorio, è necessario verificare la fruibilità delle colonnine di ricarica disponibili, al fine verificare la possibilità concreta di effettuare una ricarica programmata ovvero di individuare quella più idonea ad effettuare una ricarica imprevista.

Tali informazioni devono essere aggiornate ad intervalli regolari, affinché il sistema disponga con una certa continuità e tempestività delle informazioni necessarie ad effettuare un cambio di programma.

Per convergere su tale obiettivo, è stato ideato un sistema che è formato da due grandi unità integrate fra loro: il sistema di monitoraggio dei fattori di produzione del servizio di distribuzione merci (veicoli e colonnine di ricarica), che si compone di sensori in grado di trasferire informazioni ad una piattaforma centralizzata attraverso opportuni protocolli informatici (IoT); l'intelligenza artificiale, residente sulla stessa piattaforma che, attraverso opportuni algoritmi di calcolo (sviluppati dal DICEA della Sapienza nel corso della seconda e terza annualità di ricerca), dapprima effettua una programmazione ottimale dei giri di consegna di una flotta di veicoli elettrici per un

dato insieme di consegne da effettuare sul territorio in una giornata lavorativa e, in una seconda fase, utilizza le informazioni provenienti dal sistema di monitoraggio per restituire, in tempi adeguati a vincoli temporali stringenti, una risposta alle situazioni impreviste, senza interventi "manuali" da parte degli operatori.

Il rapporto è conseguentemente articolato in un primo capitolo che descrive l'architettura del sistema di monitoraggio dal campo e i test di funzionamento effettuati su di esso e da un secondo capitolo dedicato all'integrazione degli algoritmi di programmazione e gestione sviluppati dalla Sapienza all'interno della piattaforma eMU di ENEA.

2 Il sistema di monitoraggio di flotte veicolari elettriche

2.1 Generalità

Il problema di acquisire in tempo reale i dati da un veicolo in movimento, è stato già affrontato in passato al fine di perseguire svariati obiettivi. Spesso ad esempio, l'obiettivo è stato quello di indagare sulle emissioni del veicolo [1] [2]. Ciò ha permesso nel tempo, di mettere a punto diverse tecnologie finalizzate ad acquisire i dati in tempo reale dal veicolo.

Tutte queste tecnologie richiedono necessariamente la presenza a bordo della cosiddetta "Onboard Unit", ovvero di uno strumento finalizzato all'acquisizione dei dati. Tale strumento è in genere formato da diverse componenti, ciascuna in grado di svolgere uno specifico compito, e di interfacciarsi con le altre per scambiare dati e informazioni.

In particolare la "onboard unit" dovrà essere in grado di interfacciarsi con il CAN Bus [3] del veicolo, per scambiare messaggi al fine di acquisire le informazioni necessarie. A tal fine è opportuno utilizzare un dispositivo denominato "CAN-USB", che è in grado di mettere in collegamento il CAN BUS del veicolo con un dispositivo locale di controllo. Il dispositivo CAN-USB possiede infatti una porta seriale DB-9, che può essere collegata tramite l'apposito adattatore, alla presa OBD [4] del veicolo. Dall'altro lato invece, la porta USB di uscita del dispositivo sarà collegata al controller locale, tipicamente un PC o un Notebook su cui verrà eseguito il software per acquisire i dati di interesse. Il software in questione utilizzerà l'interfaccia software (denominata API, Application Program Interface) messa a disposizione dal dispositivo CAN-USB per comunicare con il CAN Bus ed inviare comandi di richiesta dati agli ECU del veicolo tramite il protocollo OBD. Grazie all'utilizzo della API del CAN-USB, il software è ampiamente personalizzabile, ed è possibile ad esempio scegliere quali dati acquisire, la frequenza di acquisizione, nonché la modalità di archiviazione di tali dati.

La scelta del sistema di controllo si rivela a questo punto fondamentale sia per le prestazioni, che per le possibilità di un effettivo utilizzo del sistema fuori dall'ambito prettamente prototipale o accademico. È chiaro infatti che un PC o un Notebook, sono sistemi troppo grandi e ingombranti, che non possono facilmente essere "annegati" (embedded) a bordo del veicolo per un effettivo e reale utilizzo.

Per questi motivi la nostra scelta è ricaduta sui moderni dispositivi Raspberry PI[1], sistemi di piccole dimensioni, facilmente trasportabili, ma che dal punto di vista delle prestazioni sono paragonabili a PC (o Notebook) di ultima generazione [5].

2.2 *Il sistema embedded di acquisizione dati*

La soluzione adottata prevede l'utilizzo di un sistema embedded installato a bordo del veicolo, che interagisce con il CAN Bus per acquisire i dati di interesse, elaborarli e trasmetterli ad una piattaforma remota per monitoraggio e controllo (Figura 1).

Il sistema deve essere quindi dotato di connettività GSM per la trasmissione sincrona dei dati, e di un ricevitore GPS al fine di acquisire e trasmettere anche la posizione istantanea del veicolo.

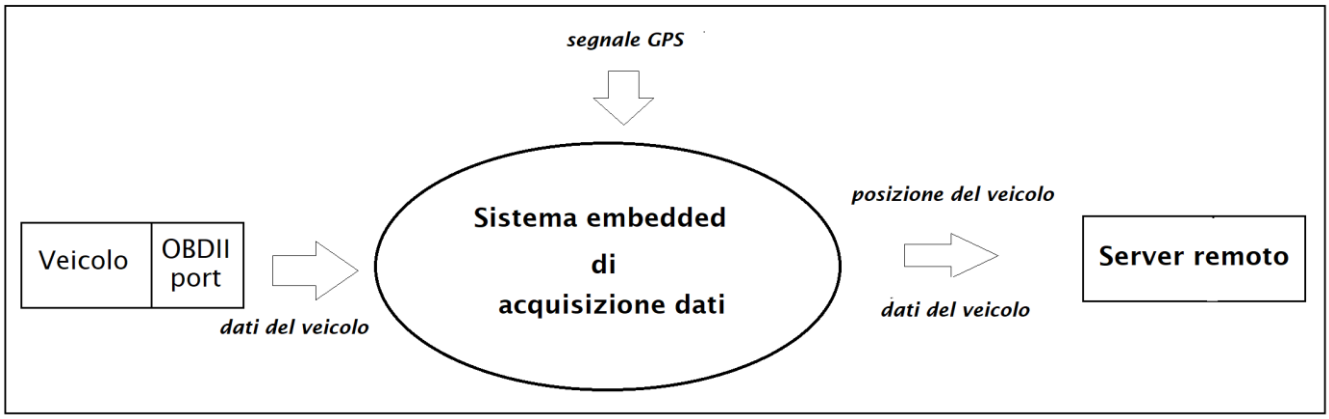


Figura 1: Il sistema embedded di acquisizione dati.

Per la realizzazione del sistema embedded si è utilizzato un dispositivo Raspberry PI 4B, dotato delle seguenti caratteristiche tecniche:

- Processore Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- 8GB LPDDR4-3200 SDRAM
- Gigabit Ethernet
- Dual Band Wireless 802.11
- Bluetooth 5.0
- 2 x USB 2.0

[1] Home Page Raspberry PI

- 2 x USB 3.0
- 2 x HDMI out port (4k)
- OS Raspbian (Linux Debian based)
- GPIO (General Purpose Input Output)

Al fine di rilevare la posizione istantanea del veicolo, al dispositivo Raspberry PI è stato connesso un ricevitore GPS tramite porta USB. La connettività a bordo è invece garantita da un modem portatile con SIM dati dedicata. Si crea in questo modo una connessione WiFi alla quale il Raspberry PI può connettersi per inviare i dati acquisiti al server remoto.

L'acquisizione dei dati dal veicolo avviene invece collegandosi alla porta OBD, attraverso una interfaccia CAN-USB. Ciò permette di portare alla porta USB del Raspberry PI i dati di interesse del veicolo, utilizzando il protocollo CAN e le API messe a disposizione dall'interfaccia CAN USB.

Il sistema completo è illustrato nello schema riportato nella seguente Figura 2.

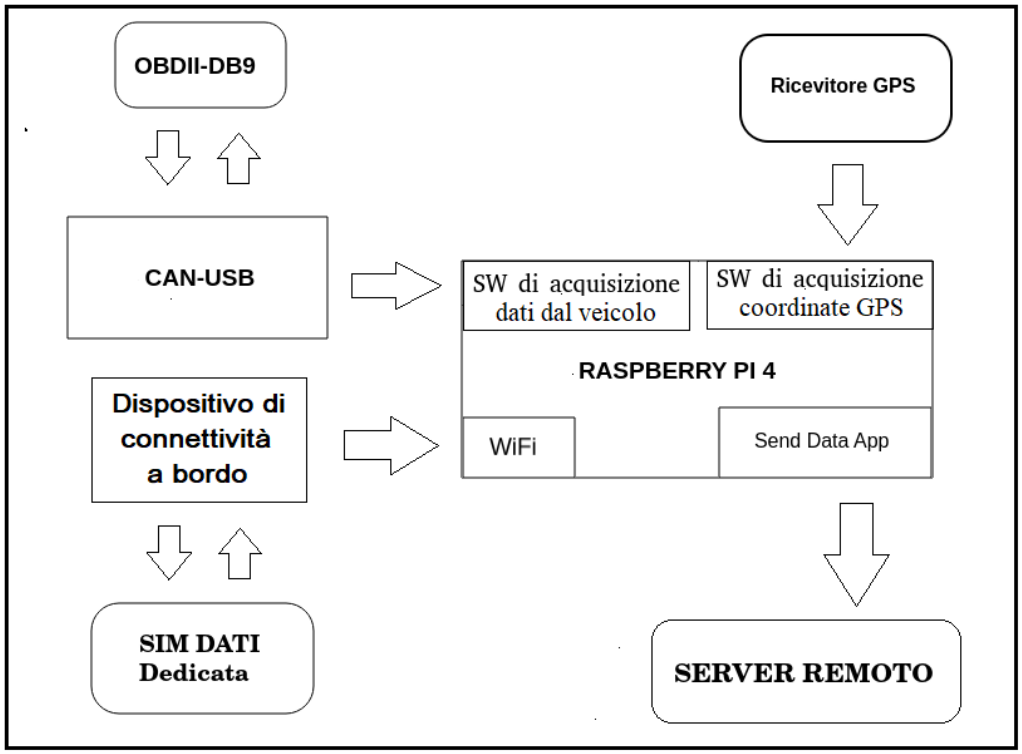


Figura 2: Gli elementi del sistema embedded.

2.3 Rilevamento della posizione del veicolo

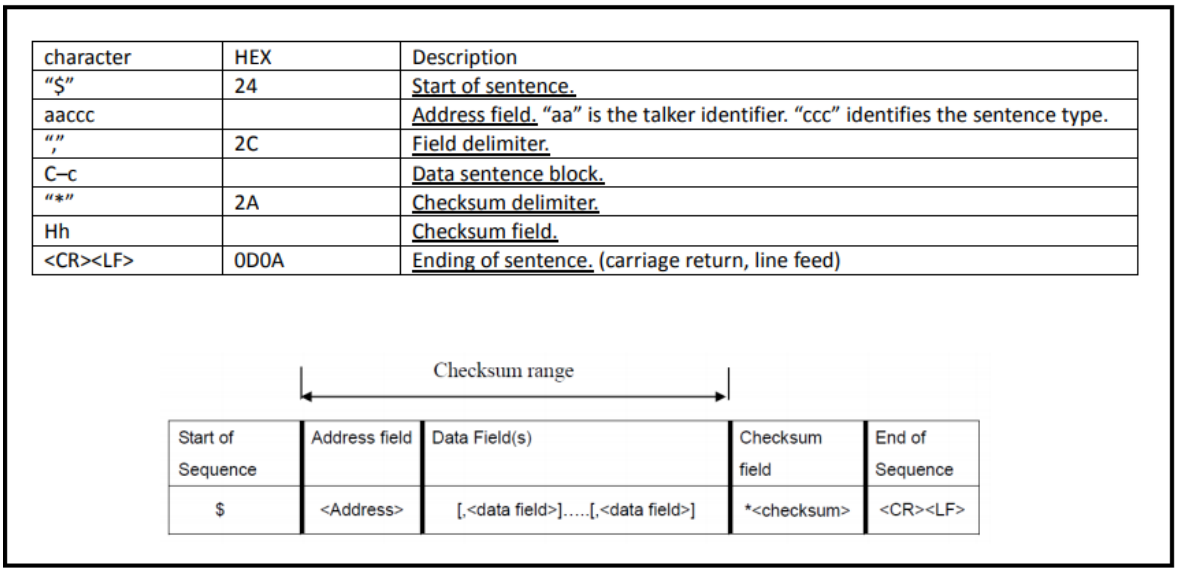
Al fine di acquisire la posizione del veicolo in movimento è stato utilizzato un rilevatore GPS collegato alla porta USB del Raspberry PI. Non occorre installare alcun software driver, il Raspberry PI rileva il dispositivo inserito e crea una interfaccia socket seriale per comunicare con esso. In Figura 3 sono visualizzati i messaggi inviati in tempo reale dal ricevitore GPS al socket seriale:

```
pi@raspberrypi:~ $ sudo cat /dev/ttyACM0
$GNRMC,120450.00,A,3807.59377,N,01319.61636,E,0.000,,180221,,,A,V*13
$GNVTG,,T,,M,0.000,N,0.000,K,A*3D
$GNGGA,120450.00,3807.59377,N,01319.61636,E,1,07,1.82,96.8,M,39.3,M,,*7F
$GNGSA,A,3,13,15,19,12,17,24,,,,,,,4.13,1.82,3.71,1*04
$GNGSA,A,3,27,,,,,,,,,,,,,4.13,1.82,3.71,4*09
$GPGSV,4,1,14,02,03,140,,06,06,106,,10,11,296,,12,56,259,22,0*64
$GPGSV,4,2,14,13,28,157,35,15,51,198,38,17,29,048,18,19,46,071,31,0*66
$GPGSV,4,3,14,20,07,262,,23,13,264,10,24,73,347,35,25,14,248,,0*6E
$GPGSV,4,4,14,28,04,055,,32,01,324,,0*6B
$GBGSV,1,1,01,27,29,153,33,0*4F
$GNGLL,3807.59377,N,01319.61636,E,120450.00,A,A*78
$GNTXT,01,01,02,u-blox AG - www.u-blox.com*4E
$GNTXT,01,01,02,HW UBX-M8130 00080000*61
$GNTXT,01,01,02,ROM CORE 3.01 (107888)*2B
$GNTXT,01,01,02,FWVER=SPG 3.01*46
$GNTXT,01,01,02,PROTVER=18.00*11
$GNTXT,01,01,02,GPS;GLO;BDS*06
$GNTXT,01,01,02,QZSS*58
$GNTXT,01,01,02,GNSS OTP=GPS;BDS*26
$GNTXT,01,01,01,NMEA unknown msg*46
$GNTXT,01,01,02,LLC=FFFFFFFF-FFFFFFFF-FFFFFFFF-FFFFFFFF-FFFFFFFD*2F
$GNTXT,01,01,02,ANTSUPERV=AC SD PDoS SR*3E
$GNTXT,01,01,02,ANTSTATUS=OK*25
```

Figura 3: Messaggi del ricevitore GPS

Si tratta di stringhe di caratteri scritte secondo lo standard NMEA 0183 (National Marine Electronics Association[2], Figura 4). Secondo tale standard, ogni stringa di caratteri è formata secondo una regola precisa e assume un altrettanto preciso contenuto informativo:

[2] Pagina web dedicata allo standard NMEA



character	HEX	Description
"$"	24	Start of sentence.
aaccc		Address field. "aa" is the talker identifier. "ccc" identifies the sentence type.
","	2C	Field delimiter.
C–c		Data sentence block.
"*"	2A	Checksum delimiter.
Hh		Checksum field.
<CR><LF>	0D0A	Ending of sentence. (carriage return, line feed)

Figura 4: Struttura della "NMEA sentence"

Il campo di checksum è un campo di controllo risultato dell'operazione di OR Esclusivo (EX-OR) tra tutti i caratteri della frase. Il valore del campo è un numero esadecimale di due cifre.

Il campo denominato "Address field" (Figura 5) invece, identifica il tipo delle informazioni che contiene quella riga di testo, i principali sono elencati nella immagine seguente:

$GPGGA	Time, position, and fix related data of the receiver.
$GPGLL	Position, time and fix status.
$GPGSA	Used to represent the ID's of satellites which are used for position fix.
$GPGSV	Satellite information about elevation, azimuth and CNR
$GPRMC	Time, date, position, course and speed data.
$GPVTG	Course and speed relative to the ground.
$GPZDA	UTC, day, month and year and time zone.

Figura 5: Address fields NMEA

2.4 Schedulazione dei processi

Al fine di gestire la schedulazione dei processi in esecuzione, è stato utilizzato il demone crond del Sistema Operativo Linux. Il demone è sempre in esecuzione, e si attiva agli intervalli regolari prefissati, per verificare all'interno del suo registro crontab, quali siano i processi da avviare. Il servizio legge il crontab[3] ed avvia l'esecuzione dei processi che sono stati schedulati per quel preciso istante temporale.

[3] Pagina web dedicata al demone crontab

2.5 Trasferimento dei dati acquisiti

Per il trasferimento dei dati rilevati verso il server di destinazione, ci si è serviti del protocollo di comunicazione SSH.

Il protocollo SSH [6] è il riferimento per le connessioni sicure, perché è in grado di stabilire una sessione cifrata tra client e server. È anche possibile scegliere l'algoritmo di crittografia da utilizzare (tra i cosiddetti algoritmi a crittografia asimmetrica) per la cifratura della trasmissione.

Un algoritmo a crittografia asimmetrica opera grazie all'utilizzo di una coppia di chiavi per ciascun client che desidera connettersi ad un server. La chiave pubblica viene condivisa con il server ed aggiunta alle authorized keys, la chiave privata viene invece custodita sul client e lo identifica univocamente. Solo l'utente in possesso della chiave privata corrispondente alla chiave pubblica conservata tra le authorized key del server, può connettersi ad esso ed effettuare le operazioni consentite dalla sessione.

Nel caso in cui il Server di destinazione, per motivi di sicurezza, sia accessibile soltanto dall'interno della rete locale LAN di appartenenza, occorre che il software di trasmissione sia in grado di connettersi alla LAN del Server prima di effettuare la trasmissione dei dati, perché altrimenti la trasmissione stessa non avrebbe successo.

Per realizzare questo tipo di connessione occorre disporre di una rete fisica dedicata allo scopo, oppure, in maniera più semplice ed economica, utilizzare una VPN (Virtual Private network).

La VPN (Virtual Private Network) è appunto una rete privata virtuale, che consente la trasmissione delle informazioni tra due reti diverse, attraverso un canale sicuro, in modo che le informazioni viaggino in forma criptata e non possano essere intercettate durante la trasmissione.

2.6 Acquisizione dati dal territorio

L'acquisizione delle informazioni dal territorio è effettuata utilizzando dei sensori magnetici da installare al suolo. Questa soluzione impiega un sensore magnetico per ciascun posto auto da presidiare. Il sensore è in grado di rilevare l'informazione di posto libero / occupato e di inviarla ad un Controller locale. Quest'ultimo è in grado di comunicare con il server remoto che acquisisce tutte le informazioni sui posti auto.

Ad intervalli regolari durante la giornata, il Server Remoto interroga il Controller per acquisire le informazioni relative a tutti i parcheggi monitorati. Il formato dell'informazione acquisita sarà tipicamente un record contenente i seguenti dati:

- Il codice numerico identificativo del posto auto esaminato
- La posizione del posto auto espressa come coppia di valori latitudine e longitudine
- La data e l'ora della rilevazione, nel formato standard Y-m-d H:M:S (4 caratteri numerici per l'anno, 2 caratteri numerici per il mese e per il giorno, ora, minuti e secondi)
- L'informazione binaria sull'occupazione o meno del posto auto.

Nella Figura 6 è riportato lo schema rappresentativo della soluzione con sensori magnetici.

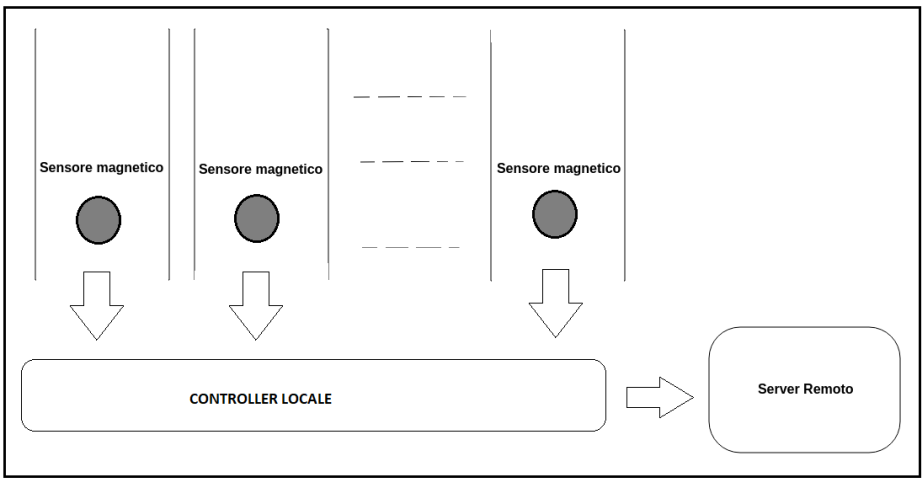


Figura 6: Soluzione con i sensori magnetici

Per i nostri scopi, al fine di realizzare un prototipo, è stato acquistato dalla ditta Banner Engineering[4], un Kit Sensore (Figura 7), composto da un sensore magnetico M-GAGE della serie DX80, ed un Controller DXM700 per il controllo del sensore. I due dispositivi comunicano tra loro tramite onde radio e l'accoppiamento tra il sensore ed il Controller avviene in maniera semi automatica all'accensione dei due dispositivi.

[4] Home Page della ditta Banner Engineering

Per avviare il processo di accoppiamento occorre infatti agire su un terzo dispositivo che agisce da sistema di attivazione tramite la sollecitazione del sensore con fascio di luce a raggi infrarossi.

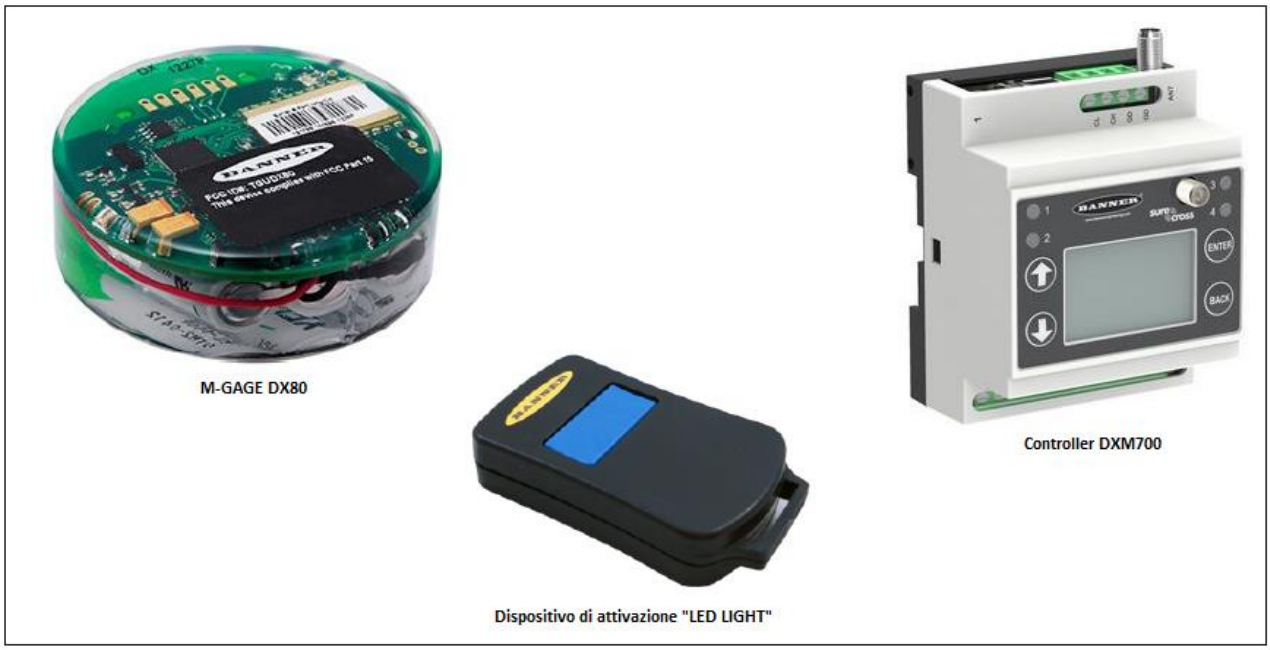


Figura 7: Il Kit Sensore della Banner Engineering

L'allegato 5.8 descrive con maggiore precisione il processo di accoppiamento tra i due dispositivi e la configurazione del sensore.

La comunicazione tra il Server remoto ed il Controller, è realizzata invece tramite il Software di gestione del Controller, denominato DXM Configuration Software, fornito dalla stessa ditta Banner Engineering.

2.7 Test sul sistema di monitoraggio di bordo

Il Raspberry PI 4 periodicamente, ad intervalli regolari, acquisisce la posizione GPS dal ricevitore, lo stato di carica della batteria, e le altre informazioni di interesse dal veicolo, ed invia tutto al server remoto. L'intervallo di rilevazione può essere variato a seconda delle applicazioni, ma dovendo in questo caso rilevare informazioni che non variano repentinamente, considerato che in un percorso cittadino il veicolo non dovrebbe mai superare la velocità media di 40 Km/h, si è pensato sufficiente fissare un intervallo di rilevazione dell'ordine di qualche minuto. Per percorsi extra-urbani è possibile aumentare ulteriormente la frequenza della rilevazione.

La trasmissione può essere effettuata a scelta, o tramite protocollo FTP, o tramite protocollo SSH per garantire un più elevato grado di sicurezza durante il trasferimento delle informazioni.

È stato anche implementato uno script di connessione VPN nel caso in cui il server remoto si trovasse all'interno di una LAN protetta da VPN.

Il Raspberry PI è alimentato dal veicolo stesso, tramite la porta accendisigari. A tal fine è stato utilizzato un alimentatore per auto che fornisca in uscita almeno la corrente di 3A. Il modem invece ha una batteria ricaricabile, la cui durata in modalità operativa è di circa 8h.

Come già accennato in precedenza, il software sviluppato per acquisire i dati del veicolo, fa uso delle API messe a disposizione dal dispositivo CAN-USB. Tutti i dispositivi commercializzati infatti, mettono a disposizione degli utenti delle librerie software già pronte, che possono essere personalizzate a seconda degli obiettivi finali. Queste librerie sono di solito scritte in linguaggio "C" che è il linguaggio di programmazione di riferimento per questo tipo di applicazioni che operano a livello del firmware dei dispositivi elettronici. Le API devono però essere personalizzate a seconda degli obiettivi, al fine di ottenere il funzionamento desiderato.

Alla libreria predefinita del dispositivo CAN-USB, che consente di leggere i codici OBD inviati sul CAN Bus, sono state quindi aggiunte diverse funzionalità, in modo da decodificare i frame OBD ed estrarre da questi le informazioni desiderate.

Le informazioni rilevate riguardano lo stato di carica della batteria (SoC), la velocità del veicolo, la velocità angolare del motore espressa in giri/minuto, la potenza fornita dalla batteria, e molte altre misure di interesse.

Per attivare il sistema occorre accendere il Raspberry PI e togliere il simbolo # di commento alle righe del crontab. Tutti i processi sono infatti gestiti dal crontab, che agli intervalli regolari prefissati, avvia la loro esecuzione. I processi per la rilevazione dei dati del veicolo e quello per la rilevazione della posizione geografica, sono avviati contemporaneamente.

Quindi, dopo un tempo di attesa di circa 60 sec, necessario affinché i processi possano terminare l'esecuzione e produrre i risultati attesi, viene avviato lo script per la trasmissione dei dati al server di riferimento.

I dati rilevati dal veicolo vengono memorizzati nel file "datalog.txt" in attesa di essere trasmessi al server di destinazione. I dati geografici invece, sono salvati nel file "gpsData.csv" in attesa di essere

trasmessi al Server di destinazione. Tutti i file inviati al server, vengono rinominati aggiungendo data e ora di invio e vengono archiviati nella directory SentData per tenere traccia del trasferimento avvenuto.

La Figura 9 mostra un esempio del file “gpsData.csv”:

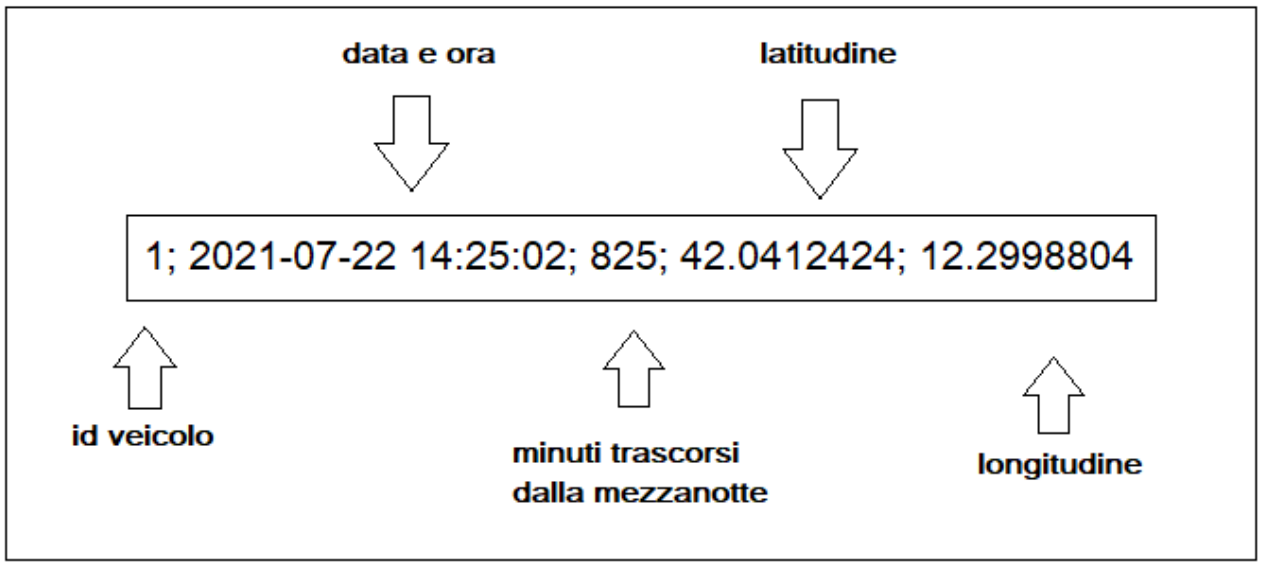


Figura 8: Esempio di dati GPS memorizzati dal sistema

Il primo valore è il codice numerico identificativo del veicolo, il secondo indica invece la data e l’ora della rilevazione. Segue quindi un campo numerico che indica i minuti trascorsi dalla mezzanotte, mentre i dati seguenti sono i valori rispettivamente di latitudine e longitudine delle coordinate spaziali.

Il file “datalog.txt” (Figura 16) contiene invece i dati rilevati dal veicolo. Nel file vengono accumulati tutti i valori dei dati di interesse durante il processo di acquisizione. Nella figura seguente mostriamo un estratto del file contenente i dati acquisiti in “forma grezza”:

```
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
20210726;165713;86.328.125;0.500000;-0.500000;392.000.000;110;10.000.000;-0.196000;0.000000;33;16;2.047.000.000;14.025.000
;;;;;;;;;;;;;
DATA;ORA;SOC;motorRPM;battA;battV;avBattery;maxBattery;battKw;vvehicle;qcvoltage;qcComm;chargerem;kwh
```

Figura 9: Il file "datalog.txt"

La Figura 10 mostra invece gli stessi dati in un formato più leggibile. I dati più interessanti che vengono rilevati sono i seguenti:

CAMPO	DESCRIZIONE	VALORE RILEVATO
Data	Data del giorno nel formato anno, mese, giorno	20210722
Ora	Orario della rilevazione nel formato ora, minuti, secondi	152706
SoC	Stato di carica della batteria espresso in percentuale	86.328125 (%)
motorRPM	Velocità angolare del motore espressa di solito in giri/minuto	0.5
battA	Corrente di carica della batteria espressa in Ampere	-0.5
battV	Tensione di carica della batteria epressa in Volt	392.0
AvBattery	Potenza disponibile della batteria (W)	110
Maxbattery	Max Battery Voltage	10
battKW	Potenza assorbita dalla batteria	0.196
Vvehicle	Velocità del veicolo	0.0 (veicolo fermo)
qcVoltage	Quick Charge Voltage	33
qcComm	Quick Charge Comm Ampere	16
chargeRem	Minuti rimanenti di durata della carica della batteria	2047
Kwh	Energia erogata dalla batteria espressa in kWh	14.025

Figura 10: I dati rilevati dal veicolo

2.8 *Test sui sensori nel territorio*

Fisicamente il Sensore magnetico è stato installato all'interno di un'area di parcheggio del Centro Ricerche ENEA della Casaccia. Il sistema così realizzato funziona da prototipo, ma è ampiamente scalabile, perché un Controller può gestire diverse decine di sensori, disposti fisicamente nell'arco di circa 30 m di distanza dal Controller, senza ostacoli intermedi.

I dati di interesse in questo caso sono i seguenti:

- Il codice identificativo del sensore, che lo identifica in maniera univoca
- La posizione del sensore espressa in coordinate di latitudine e longitudine. Essendo il sensore fisso, la sua posizione sarà rilevata una sola volta al momento dell'installazione e poi non sarà più modificata
- La data e l'orario della rilevazione
- L'informazione binaria sullo stato di occupazione del posto auto (libero/occupato)

Nella Figura 11 è riportato un esempio di record che può essere ottenuto in rilevazione:

codice ID	latitudine	longitudine	data e ora	busy
0001	42.0412424	12.2998804	2021-07-16 11:05:32	True/False

Figura 11: Il record delle informazioni rilevate

Il campo codice ID contiene il codice identificativo del sensore, latitudine e longitudine indicano la sua posizione, quindi segue la data e l'ora della rilevazione, ed infine lo stato di libero/occupato del parcheggio presidiato dal sensore.

Gli intervalli di rilevazione possono essere regolati a piacimento, un valore tipico potrebbe essere dell'ordine di qualche minuto.

3 Il sistema di programmazione e gestione delle consegne urbane con flotte elettriche

3.1 Architettura generale

Il problema dell'ottimizzazione dei percorsi per le consegne di merci nell'ambito di una realtà locale è un argomento di vasto interesse sia accademico che operativo, basti pensare alle migliaia di consegne che ogni giorno vengono gestite nell'ambito dell'e-commerce. Sistemi di ottimizzazione e di gestione delle deviazioni dalle tabelle di marcia programmate sono oggi utilizzati da ogni vettore commerciale coinvolto nelle consegne di merci [7] [8] [9]. La quasi totalità delle tecnologie veicolari utilizzate, tuttavia, è caratterizzata da mezzi con motore termico, mentre poco o addirittura assenti sono le analisi, simulazioni e sperimentazioni relative alle peculiarità legate all'introduzione di veicoli elettrici.

Se si considerano veicoli a trazione elettrica, a fianco delle normali problematiche di gestione dei percorsi, del traffico, delle consegne, è necessario aggiungere il trattamento del livello di carica e di eventuali ricariche presso le colonnine disponibili. Questo porta ad un aumento delle possibili criticità, in particolare legato alla necessità di programmare una ricarica all'interno del percorso previsto.

Il sistema realizzato in questo lavoro considera le consegne giornaliere da effettuare su un'area urbana che comprende larga parte della città metropolitana di Roma. Una flotta composta da mezzi elettrici di diversa capacità di carico e diversa autonomia viene utilizzata per ottimizzare la consegna delle merci all'interno di finestre temporali associate ad ogni consegna. L'ottimizzazione, descritta nel seguito e più in dettaglio nelle relazioni della Sapienza (RdS/PTR(2020)/061 e RdS/PTR(2021)/204), permette di programmare le consegne del giorno utilizzando una funzione obiettivo che considera e ottimizza il numero di mezzi impiegati, il numero totale di km percorsi e il tempo totale impiegato.

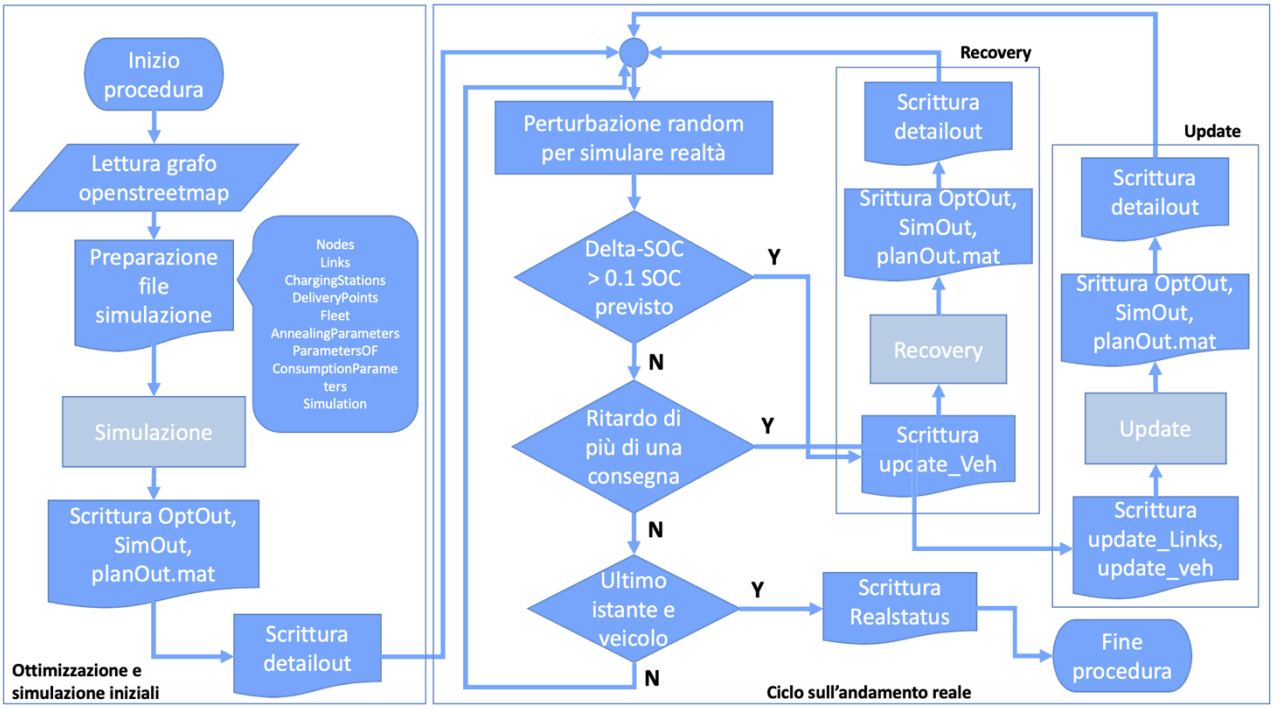


Figura 12: Schema della procedura utilizzata per l'ottimizzazione dei percorsi iniziale e per la gestione dei percorsi reali, con il lancio eventuale dei processi di recovery e di update.

Nel diagramma di flusso presentato nella precedente figura, i processi di ottimizzazione iniziale dei giri di consegna giornalieri e quelli di gestione delle eventuali anomalie in corso d'opera sono realizzati dai codici di 'Simulazione', 'Recovery' ed 'Update' sviluppati in Matlab dalla Sapienza e disponibili in ambiente eMU come eseguibili compilati per sistema operativo Linux.

Il processo di 'Simulazione', oltre a garantire una programmazione dei giri di consegna affrontando un classico problema di Vehicle Routing con Time-Windows, affronta anche il vincolo di autonomia dei veicoli elettrici, prevedendo la possibilità di effettuare ricariche presso le stazioni di ricarica disponibili sul territorio, nel caso i percorsi previsti lo rendano necessario. Il codice di 'Recovery', invece, gestisce eventuali situazioni di default della batteria dei veicoli elettrici, suggerendo ricariche aggiuntive o alternative rispetto a quelle programmate. Il codice di 'Update', infine, gestisce situazioni di ritardo del veicolo senza che questo comporti necessariamente un'allerta sull'autonomia residua. Una descrizione approfondita degli algoritmi di programmazione ottimale delle consegne e di gestione in tempo reale delle anomalie sono contenuti nelle relazioni complementari alla presente redatte dalla Sapienza (RdS/PTR(2020)/061 e RdS/PTR(2021)/204)

Le procedure accessorie per la creazione dell'ambiente di lavoro e dei file di input dei codici di della Sapienza sono invece state sviluppate da ENEA in ambiente python.

L'intera procedura è stata pianificata e realizzata in modo da evitare l'intervento dell'operatore una volta che la suite è stata lanciata.

3.2 Funzione di programmazione off-line

I dati di input del processo di programmazione possono essere divisi in due parti, legate rispettivamente ai parametri per il funzionamento delle procedure di Ottimizzazione/Simulazione (coefficienti, soglie, …) e ai dati dell'istanza da processare, in particolare relativi alla rete stradale (archi e nodi del grafo), ai veicoli presenti nella flotta, alle consegne da effettuare e alle stazioni di ricarica disponibili sul territorio.

La struttura di tali dati, scambiati in formato csv, è riportata nelle Tabelle da 1 a 9 di esempio. Il formato dati di input e di otuput è stato concordato con la Sapienza al fine di integrare il codice di calcolo nella piattaforma eMU e allo stesso tempo per garantire che l'intero processo possa essere gestito in modo completamente automatico.

Tabella 1: Esempio del file di input AnnealingParameters.csv, finalizzato alla valorizzazione dei parametri utilizzati all'interno dell'algoritmo di annealing, cuore del processo di ottimizzazione/simulazione

T0	Tend	L	Q
1	0.0001	500.0	0.95

Tabella 2: Esempio del file di input ParametersOF, finalizzato alla valorizzazione dei parametri utilizzati all'interno della funzione obiettivo dell'ottimizzazione per stabilire i pesi relativi fra il numero di veicoli, i km percorsi e il tempo totale necessario per le consegne

km_cost [€cent/km]	h_cost [€cent/h]	Veh_cost [€cent/veicolo]	beta1	beta2	beta3	beta4
0.05	0.5	200.0	100	1	1	10

Tabella 3: Esempio del file di input Simulation.csv, necessario per fornire al modulo di simulazione i parametri utilizzati per limitare l'estensione e la frequenza della simulazione

Date [YYYYMMDD]	simulation Start [min]	simulation End [min]	timeStep [min]
20210325	480	1200	5

Tabella 4: Struttura del file di input DeliveryPoints.csv, finalizzato alla localizzazione dei punti di consegna e delle consegne ad essi associate

Delivery Point ID	Delivery point longitude	Delivery point latitude	Demand_kg	Demand_m3	Time Window Start [minuti a partire 0:00]	Time Window End [minuti a partire 0:00]	Date [YYYYMMDD]
25418888	12.4912621	41.89958529	0	0.0	0	0	20210325
251790652	12.4772305	41.88126029	43	0.14	480	780	20210325

Tabella 5: Esempio del file di input Fleet.csv, finalizzato alla descrizione della flotta disponibile e delle sue caratteristiche, da cui saranno poi selezionati i veicoli realmente utilizzati per le consegne

VehicleID	VehicleType	maxVehicleWeight [kg]	maxCharge [kWh]	VehicleLoadCap [kg]	VehicleVolumeCapacity [m3]	VehicleLoadTime [minutes]	DeliveryUnloadTime [minutes]	avg_cons [kWh/km]
1598	Mini	1800.0	25.0	450.0	5.5	15.0	5.0	0.2
1654	Small	3500.0	30.0	1100.0	15.0	20.0	5.0	0.38
6184	Midi	7500.0	50.0	2550.0	30.0	25.0	5.0	0.8

I valori del consumo chilometrico di riferimento riportati nell'ultima colonna della precedente tabella sono stati stimati da ENEA per un valore di velocità pari a 30 km/h, per un carico medio del 50 %, utilizzando le curve di consumo illustrate nella successiva Figura 13. Tali valori di riferimento sono utilizzati dal modulo di Ottimizzazione dei giri di consegna sviluppato dalla Sapienza che rappresenta il primo step del processo di simulazione.

Tabella 6: Struttura del file di input ChargingStations.csv, finalizzato alla localizzazione delle colonnine di ricarica disponibili sul territorio considerato

ChargingStation ID	Charging Station Longitude	Charging Station Latitude
382501377	12.324249	41.80673
388196589	12.2801525	41.7309823

Per le colonnine di ricarica sparse sul territorio viene assunto un valore di riferimento della potenza pari a 150 kW al fine di consentire operazioni di ricarica rapida per tutte le classi dimensionali dei veicoli considerate. Con tale elevato valore di potenza si intende assicurare tempi di ricarica durante il servizio mai superiori a 30 minuti. Naturalmente, per la ricarica notturna al deposito si assumono valori di potenza molto più bassi, intorno ai 25kW, adatti ad effettuare una ricarica di tipo lento, considerata una disponibilità di tempo intorno alle 8 ore al minimo.

Tabella 7: Struttura del file di input Nodes.csv, contenente le coordinate dei nodi del grafo considerato dalla simulazione

ID_Node	xcoord	ycoord
1474297856	11.8328285	42.0400933
959447048	12.5347745	41.5675739
959447049	12.5401895	41.5679398
959447052	12.5193668	41.5693023

Tabella 8: Struttura del file di input Links.csv, contenente gli archi del grafo considerato dalla simulazione. I nodi di inizio e fine di ogni arco sono contenuti nel file Nodes.csv, inoltre le velocità sono variabili in funzione dell'ora della giornata

StartNodes	EndNodes	Length [m]	ID	Speed (00:00-00-59) [km/h]	Speed (01:00-01-59) [km/h]	Speed (02:00-02-59) [km/h]	Speed (03:00-03-59) [km/h]	Speed (… - …)
1474297856	1474254970	71.929	1	21	21	21	21	…
959447048	959447006	80.352	2	21	21	21	21	…
959447048	5473407351	83.949	3	21	21	21	21	…
959447048	959447926	50.743	4	21	21	21	21	…

Tabella 9: Esempio del file di input ConsumptionParameters.csv, contente i coefficienti utilizzati all'interno della Simulazione per calcolare i consumi dei mezzi della flotta. Il tipo di veicolo è associato a quello del file Fleet.csv

Veicle type	Load %	a	b	c
Midi	0	0.000124	-0.019388	1.162513
Midi	50	0.000156	-0.024381	1.461949
Midi	100	0.000187	-0.029375	1.761384
Small	0	0.00006	-0.0094	0.563643
Small	50	0.000074	-0.011554	0.692811
Small	100	0.000088	-0.013708	0.821979
Mini	0	0.000034	-0.005288	0.317049
Mini	50	0.000039	-0.006169	0.369891
Mini	100	0.000045	-0.00705	-0.422732

La precedente tabella riporta i valori dei coefficienti delle curve di consumo utilizzati dalla Sapienza per il calcolo dei consumi dei furgoni elettrici nel modulo di Simulazione. Tali valori sono stati estrapolati da ENEA da misure ottenute attraverso campagne di rilevamento effettuate sul campo nel precedente triennio di ricerca 2015-2018 per bus elettrici a batteria. Nella Figura 13 sono riportate le curve di consumo in funzione delle velocità e del carico del veicolo per le tre tipologie di furgone prese in considerazione per la distribuzione urbana della merci. Le curve sono state calcolate per una pendenza media pari al 2% tenuto conto dei sistemi di recupero dell'energia di frenata di cui si assume che i veicoli siano dotati. Le equazioni delle curve di consumo, interpolate sui valori sperimentali, sono delle polinomiali di secondo grado, in cui a è il coefficiente del termine quadratico, b del termine lineare e c è la costante.

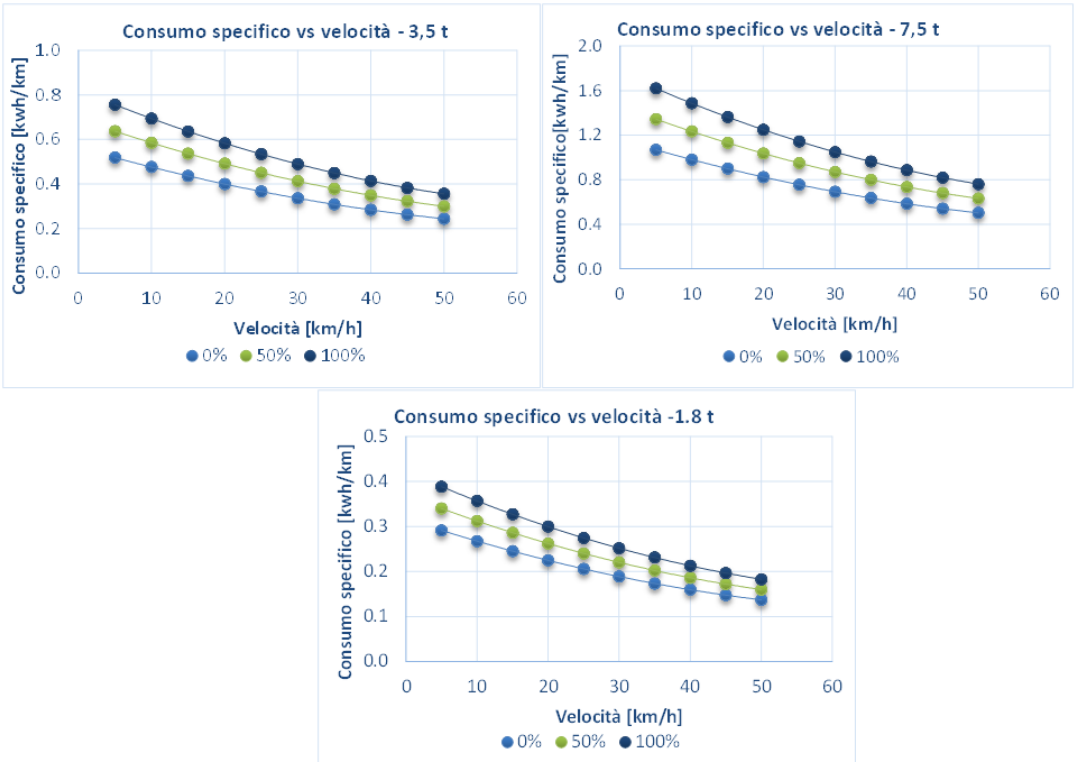


Figura 13: Curve di consumo per i veicoli merci da 1.8t, 3.5t e 7.5t al variare del carico (0%, 50%, 100%)

Il calcolo dei giri ottimali dei veicoli presenti nella flotta per un dato insieme di consegne si basa su una preliminare elaborazione delle matrici delle distanze e dei tempi di percorrenza fra i punti di consegna coinvolti e tra i punti di consegna e le stazioni di ricarica, che dipende dal grafo utilizzato. Per questo motivo prima del lancio dell'ottimizzazione, si è reso necessario implementare una procedura, sviluppata da ENEA, che caricasse il grafo dell'area considerata contenuto nella piattaforma eMU. In particolare eMU utilizza il grafo opensource "openstreetmap" [10], arricchito delle necessarie informazioni sui tempi di percorrenza e sulle velocità associate agli archi e infine salvato nei file di input che descrivono i nodi e gli archi.

L'ottimizzazione e simulazione iniziali producono un primo insieme di ouput che vengono denominati con il suffisso 'orig' e che costituiscono il riferimento per il set di consegne affidate ad ogni veicolo ed il relativo percorso.

Più nel dettaglio, il processo di ottimizzazione produce in output un primo file OptOut.csv (Tabella 10) contenente una riga per ogni fermata del veicolo, compresa la partenza e il ritorno al deposito e le eventuali fermate alle stazioni di ricarica. Le informazioni associate riguardano l'orario di arrivo e partenza previsto, la distanza percorsa dalla partenza dal punto precedente, il peso della

consegna da effettuare, il carico del veicolo in partenza (una volta effettuata la consegna), lo Stato di Carica (SOC) della batteria espresso in kWh alla ripartenza.

Tabella 10: Esempio di un file OptOut.csv in uscita dal software 'Simulazione'

VehicleID	DeliveryPointID	ArrivalTime [minuti a partire 0:00]	DepartureTime [minuti a partire 0:00]	TravelledDistance [km]	Deliveries	VehicleLoad [kg]	SOC [kWh]
1598	25418888	0	478	0	0	499.4	25
1598	2140000000	480	485	0.747	36	463.4	25
1598	25418915	486	491	1	90	373.4	25
1598	1110000000	492	497	2	24.6	348.8	25
1598	292000000	503	508	5	25.4	323.4	24
1598	259000000	509	514	5	36	287.4	24
1598	296000000	518	523	8	71	216.4	23
1598	302000000	528	533	10	18.8	197.6	23
1598	1140000000	538	543	13	15.6	182	22
1598	302000000	548	553	16	20	162	22
1598	337000000	556	561	17	34.8	127.2	22
1598	258000000	563	568	19	41.6	86	21
1598	246000000	569	574	19	35.6	50	21
1598	255000000	576	581	20	50	0	21
1598	25418888	584	0	22	0	0	21

Un secondo output, SimOut.csv (Tabella 11), è invece composto da una sequenza di stati del veicolo con frequenza fissa di 5 minuti, con informazioni simili alle precedenti ma non più associabili ad un punto di consegna o di ricarica ma piuttosto ad un nodo generico del grafo utilizzato; in particolare sono riportati le distanze e i tempi di percorrenza parziali e totali in arrivo per ogni istante considerato e la velocità media di percorrenza.

Tabella 11: Esempio di un file simOut.csv in uscita dal software 'Simulazione'

Vehicle	SimTime	Latitude	Longitude	Node	SOC [kWh]	Dist [km]	Diffdist [km]	Difftime [minuti]	Velocity [km/h]
1598	475	41.8995853	12.4912621	25418888	25	0	0	0	NaN
1598	480	41.8972323	12.4958768	2141800773	25.43128927	0.746874	0.746874	1.697848	26.39
1598	485	41.9005798	12.5003203	25418915	25.43128927	0.746874	0	0	NaN
1598	490	41.9005798	12.5003203	25418915	25.74890522	1.266992	0.520118	0.916170286	34.06
1598	495	41.9021397	12.4941265	1107360243	26.15907963	1.944564	0.677572	1.241307429	32.75
1598	500	41.9081416	12.503612	276549264	26.95721222	3.301717	1.357153	2.835210857	28.72
1598	505	41.9063534	12.5165416	291633262	27.87406441	4.86255	1.560833	3.278102286	28.56
1598	510	41.9096362	12.5212302	259031441	28.22858021	5.442048	0.579498	1.012516571	34.34
1598	515	41.9084938	12.525185	259031451	28.14530769	5.854797	0.412749	0.707569714	35
1598	520	41.9004505	12.534988	295784322	27.7972398	7.580032	1.725235	2.957545714	35

Vehicle	SimTime	Latitude	Longitude	Node	SOC [kWh]	Dist [km]	Diffdist [km]	Difftime [minuti]	Velocity [km/h]
1598	525	41.89546	12.5413745	1537573800	27.61990437	8.376382	0.79635	1.631681143	29.28
1598	530	41.8790152	12.5440592	301687816	27.20536336	10.431098	2.054716	3.522370286	35
1598	535	41.8766733	12.5530211	301687270	27.03728632	11.253637	0.822539	1.440789143	34.25
1598	540	41.862157	12.5530531	1140901717	26.62236566	13.170101	1.916464	3.721185143	30.90
1598	545	41.8639819	12.5479018	299620430	26.47856524	13.817073	0.646972	1.320749714	29.39
1598	550	41.8684365	12.5310163	302184215	26.08971823	15.744434	1.927361	3.304047429	35
1598	555	41.8760532	12.5215696	302175449	25.84385498	16.963081	1.218647	2.089109143	35
1598	560	41.8782034	12.5186356	336604863	25.74547251	17.450724	0.487643	0.835959429	35
1598	565	41.8814548	12.5091728	258207298	25.53204823	18.679992	1.229268	2.107316571	35
1598	570	41.884417	12.5111212	246118245	25.45001933	19.152457	0.472465	0.80994	35
1598	575	41.8871824	12.5046011	1672413680	25.33292616	19.826883	0.674426	1.156158857	35
1598	580	41.8915656	12.502157	255259856	25.24141645	20.353955	0.527072	0.903552	35
1598	585	41.8995853	12.4912621	25418888	24.91226083	22.101114	1.747159	3.472385714	30.18

Per consentire un confronto fra la programmazione iniziale dei giri di consegna ed il monitoraggio continuo previsto sul campo, è stato ritenuto opportuno intensificare notevolmente la frequenza della previsione dello stato del veicolo da 5 minuti a 10 secondi. Questa operazione è svolta da ENEA in fase di postprocessing per mezzo di codice sviluppato in linguaggio python a partire dalle informazioni contenute in OptOut.csv e SimOut.csv. Non si tratta tuttavia di una mera interpolazione delle informazioni: per il calcolo delle traiettorie tra due punti successivi nel file SimOut.csv viene infatti ottimizzato il percorso mediante l'utilizzo del grafo openstreetmap. Il risultato viene salvato nel file DetailSim.csv, la cui struttura è mostrata in Tabella 12, e che contiene informazioni sia sulla posizione dei veicoli, sia sul loro stato e sullo stato delle spedizioni.

Tabella 12: Struttura di un file Detailsim.csv elaborato a partire dai file OptOut.csv e simOut.csv. Il file contiene una riga ogni 10 secondi per ogni mezzo, solo le prime righe sono mostrate

Vehicle	SimTime	Latitude	Longitude	Node	SOC [kWh]	Dist [km]	Delivered
1654	28500	41,8995853	12,4912621	25418888	30	0	0
1654	28510	41,8995853	12,4912621	25418888	30	0	0
1654	28520	41,8996964	12,491662	25418888	30	0	0
1654	28530	41,8996964	12,491662	25418888	30	0	0
1654	28540	41,8991009	12,4923408	254528047	29,95485983	0	0
1654	28550	41,8991009	12,4923408	254528047	29,95485983	0	0
1654	28560	41,8996911	12,4931818	254528048	29,90604308	0	0
1654	28570	41,8996911	12,4931818	254528048	29,90604308	0	0
1654	28580	41,9001763	12,4939276	254528049	29,862753	0	0

Vehicle	SimTime	Latitude	Longitude	Node	SOC [kWh]	Dist [km]	Delivered
1654	28590	41,8996708	12,4948262	246150307	29,813819	0,093223	0
1654	28600	41,8991423	12,4957656	252608212	29,77531423	0,190683	0
1654	28610	41,8990232	12,4959605	2126941296	29,76200672	0,211554	0
1654	28620	41,8990232	12,4959605	2126941296	29,76200672	0,211554	0
1654	28630	41,8982896	12,4969903	669160655	29,70718284	0,290903	0
1654	28640	41,8982896	12,4969903	669160655	29,70718284	0,290903	0
1654	28650	41,8981663	12,4968832	669174322	29,68567799	0,290903	0
1654	28660	41,8976282	12,4975274	4286174885	29,65365889	0,371047	0
1654	28670	41,897344	12,4978677	252608902	29,62507909	0,413379	0

L'insieme dei tre file OptOut.csv, simOut.csv e Detailsim.csv contiene tutte le informazioni necessarie per visualizzare nella piattaforma eMU i giri di consegna inizialmente previsti. In Figura 14 è mostrato un esempio di visualizzazione del risultato di una ottimizzazione che coinvolge 4 veicoli di diversa capacità per la consegna di 124 pacchi nell'area di Roma Capitale. Risulta evidente che le consegne vengono suddivise per aree in modo da ridurre al minimo il numero di veicoli utilizzati, il numero di km percorsi e il tempo totale impiegato.

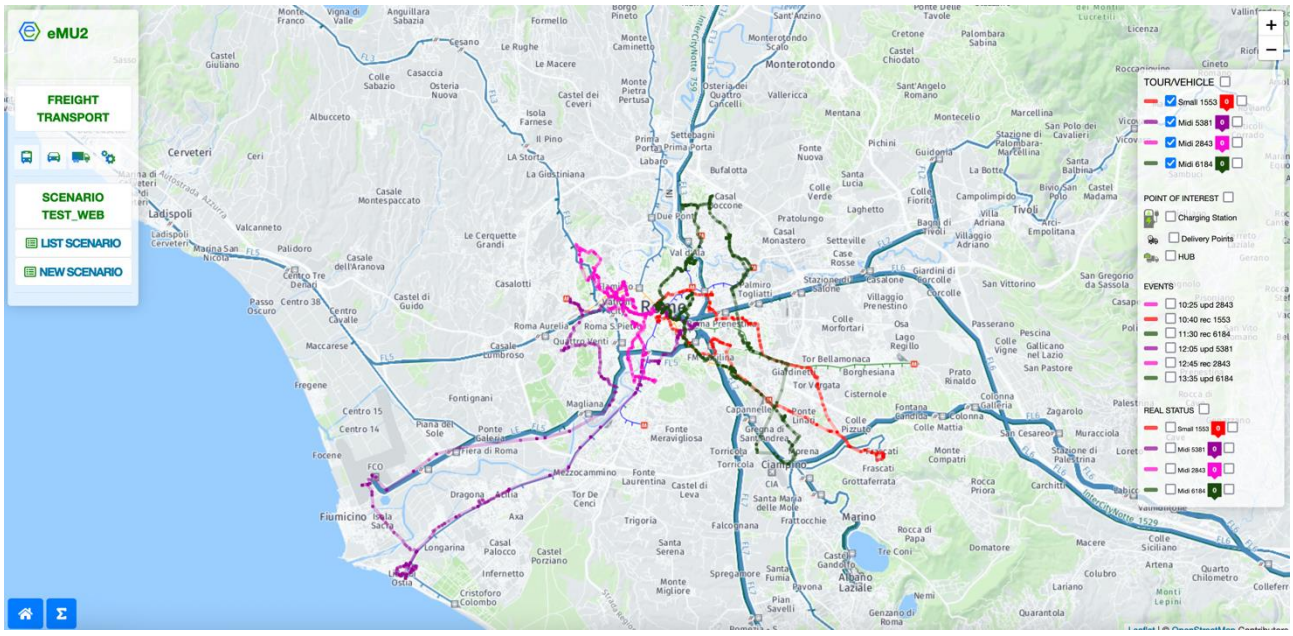


Figura 14: Esempio di output di una ottimizzazione e simulazione iniziali, in cui sono coinvolti 4 veicoli, di cui 3 di taglia midi e uno di taglia mini

3.3 Funzione di gestione in tempo reale delle anomalie

In fase di esercizio, la procedura è pensata per acquisire in tempo reale i dati dai veicoli sul territorio, con frequenza di 10 secondi, che sono salvati nel file Realstatus.csv, contenente lo stato

di tutti i veicoli ogni 10 secondi e la cui struttura è analoga a quella del file Detailsim.csv in Tabella 12.

Il confronto fra i dati reali e quelli previsti (contenuti in Detailsim.csv) rappresenta il motore delle scelte del sistema durante la giornata di lavoro. I criteri con cui effettuare tale confronto ed attivare, eventualmente, delle azioni di recupero, sono gestiti attraverso un ulteriore file di input modificabile dall'utente (vedi successiva tabella), che attiene specificatamente alla fase di esercizio della flotta.

Tabella 13: Esempio del file di input Parameters.csv, necessario per fornire al software di controllo (python) i parametri di soglia per le scelte di simulazione (lancio di recovery e update, ecc.)

mysimoutinterval	random_delay	random_velocity	randthres	SOC10s	velreduction	socratio	Socabs	deldelay
10	10	4	0.8	0.01	0.7	0.05	0.4	3.0

Quando la distanza fra le informazioni contenute nei due file Realstatus.csv e Detailsim.csv supera le soglie fissate vengono automaticamente creati i file di input per le azioni di Recovery o Update.

Queste procedure possono essere lanciate più volte per lo stesso mezzo, in modo da gestire ogni criticità durante il percorso.

Al termine del processo, il codice produce file analoghi a quelli della simulazione iniziale, esclusivamente per il veicolo per il quale si è rilevata l'anomalia e non per tutti i veicoli della flotta. Viene anche creato un nuovo file binario planOut.mat, in modo da permettere l'eventuale lancio di un nuovo recovery o update nel caso le condizioni reali del mezzo lo richiedano.

Alla fine della giornata l'utente ha a disposizione i file per ogni update o recovery utilizzato, contrassegnati nel nome con l'identificativo del mezzo e con l'istante di lancio.

3.4 Test del sistema

La verifica di efficacia delle soluzioni di ottimizzazione dei giri di consegna su un caso di test realistico è stata effettuata della Sapienza, al cui Report si rimanda.

Per il testing delle funzionalità di recupero delle anomalie, non potendo per ovvi motivi disporre di un sistema di monitoraggio completo sul campo, lo stato reale dei veicoli è stato simulato da ENEA attraverso l'introduzione di perturbazioni casuali sul sistema simulato.

A partire dai percorsi e dallo stato del veicolo previsti, ogni 10 secondi è stato valutato con un processo random (con probabilità del 20%, regolabile in fase di lancio del test) se introdurre o meno un ritardo di 10 secondi per il veicolo e contemporaneamente se introdurre o meno un incremento di utilizzo della batteria di 0.01 kWh. Di conseguenza si è potuto osservare un progressivo aumento del ritardo e della differenza di carica del SOC rispetto a quanto previsto, man mano che il percorso dei mezzi avanzava. Nell'esperienza reale, gli scostamenti rispetto alla simulazione possono essere sia peggiorativi sia migliorativi, nel test sono stati imposti solamente scostamenti peggiorativi, in modo da mettere sotto stress il sistema modellistico.

La procedura monitora separatamente lo scostamento rispetto alla previsione della carica e del numero di consegne effettuate, attivandosi in modo differente a seconda dei casi.

Il controllo sullo stato di carica della batteria viene eseguito ogni 5 minuti (in modo da mantenere la compatibilità con la frequenza di restituzione della simulazione originale). Viene monitorato sia il livello assoluto della carica, sia lo scostamento rispetto alla carica prevista, in modo da evitare di mandare in ricarica un veicolo con un basso livello del SOC, che tuttavia si trova nella fase di rientro al deposito e che quindi non necessita di essere ricaricato prima della fine del giro di consegne.

Nel test, la ricarica è stata artificialemente attivata solo se il SOC fosse risultato inferiore al 20% della capacità totale e se, contemporaneamente, la differenza di carica reale rispetto a quella prevista fosse stata superiore al 5% della carica prevista (parametri comunque modificabili dall'operatore).

Quando si sono verificate tali condizioni, il sistema ha correttamente predisposto il file di input Update_Veh.csv, mostrato in Tabella 14, per il lancio del modulo di 'Recovery', la parte del codice finalizzata a programmare una nuova ricarica per il veicolo.

Tabella 14: Esempio di un file Update_Veh.csv necessario per il lancio delle procedure di recovery e update

VehicleID	timeStamp [minuti dalle 0:00]	Latitude	Longitude	Node	SOC [kWh]	Dist [km]	lastDel
6184	815.0	41.9312579	12.5185195	1678480002	32.3374418888574	65.221609	27

Questo file è stato letto dal codice di Recovery insieme alle uscite binarie della simulazione iniziale o, se esistenti, dei precedenti recovery o update. Il codice ha correttamente annullato tutte le

ricariche precedentemente pianificate per il veicolo e riprogrammato altre ricariche, a partire da quella presso la stazione più vicina, per poi far tornare il veicolo sul tragitto previsto, in modo da effettuare le consegne ancora mancanti.

In Figura 15 è mostrato un esempio di modifica del percorso associato ad impreviste necessità di ricarica della batteria. Le condizioni per il lancio della procedura di recovery si verificano subito dopo la consegna al punto 11, dopo la quale il veicolo viene deviato verso la stazione di ricarica più vicina, effettua la nuova ricarica e viene successivamente riportato sul percorso originare per effettuare la consegna successiva n. 12 e continuare il giro.

Nella Figura 15 è riportato un esempio di recovery per uno dei mezzi della flotta. Il recovery interessa il veicolo a partire dall'undicesima consegna, suggerisce una deviazione per la ricarica e successivamente rimette il veicolo sul percorso precedente per effettuare le consegne successive. Il percorso previsto dal recovery è segnato con tratto più marcato, il percorso dell'ottimizzazione iniziale con tratto semitrasparente.

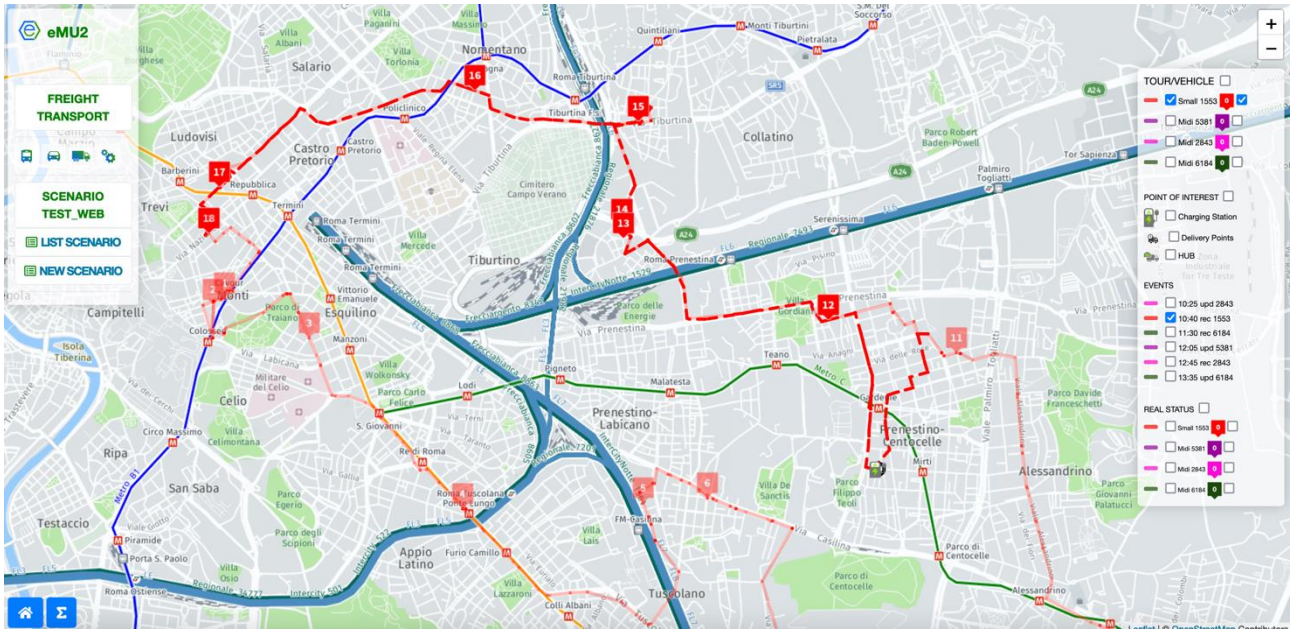


Figura 15: Esempio di un recovery

Per testare la funzionalità dell'Update, ovvero della procedura che si attiva quando il veicolo accumula un ritardo eccessivo rispetto a quanto programmato, considerando anche eventuali modifiche delle condizioni di congestione della rete, è stato imposto che non si potesse

accumulare (artificialmente, tramite procedura casuale) un ritardo di più di 3 consegne (parametro modificabile dall'operatore). Anche in questo caso, i percorsi previsti dalla precedente simulazione sono stati regolarmente annullati e riprogrammati utilizzando le velocità aggiornate degli archi del grafo. Inoltre, nei casi dove questo fosse necessario, sono state previste le soste per la ricarica nell'ambito del nuovo percorso necessarie per poter concludere il giro di consegne.

Tabella 15: Struttura di un file Update_Links.csv necessario per il lancio delle procedure di update. L'ID è relativo all'arco del grafo, mentre le velocità sono quelle disponibili in tempo reale

ID	Speed [km/h]
1	28
2	28
3	11
4	38

Il lancio dell'update condivide con il recovery il primo file di input, Update_Veh.csv, a cui viene aggiunto un secondo file Update_Links.csv, un cui esempio è riportato in Figura 15 finalizzato all'aggiornamento del grafo e delle velocità associate agli archi con le più recenti informazioni in tempo reale. Anche l'update produce i file di output del recovery, in modo da creare le condizioni per un eventuale nuovo recovery o update nel caso il successivo andamento reale del veicolo lo richieda.

In Figura 16 è mostrato un esempio di update relativo ad uno dei mezzi della flotta. L'update interessa il veicolo a partire dalla diciottesima consegna, il percorso viene radicalmente modificato in base alle velocità degli archi aggiornate in tempo reale. Il percorso previsto dall'update è segnato con tratto più marcato, il percorso dell'ottimizzazione iniziale con tratto semitrasparente

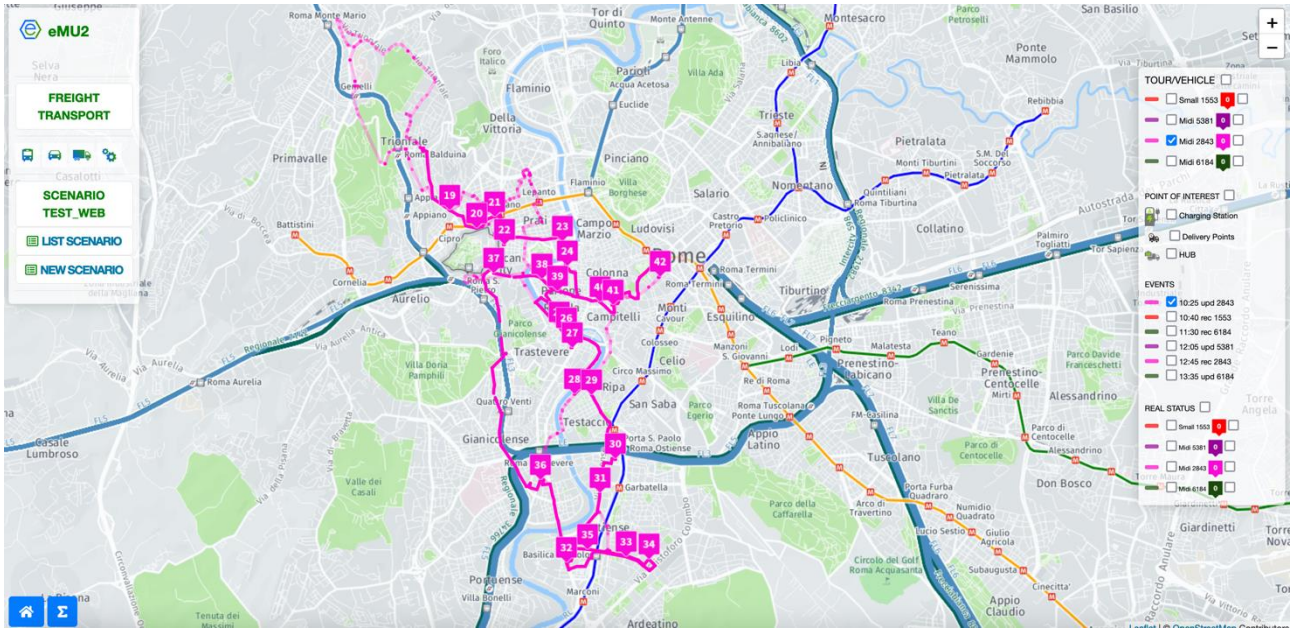


Figura 16: Esempio di un update

3.5 Istruzioni per l'utilizzo della piattaforma eMU

Il sistema di ottimizzazione e gestione dei giri di consegna è stato integrato nella piattaforma web-based eMU, un DSS multifunzione ideato e sviluppato da ENEA per favorire la diffusione della mobilità elettrica nelle aree urbane, con mezzi privati, pubblici e furgoni merci.

All'interno di eMU sono state create apposite interfacce grafiche dedicate alle funzionalità specifiche per la distribuzione delle merci che ne rendono semplice ed intuitivo l'impiego da parte dell'utente finale.

Dalla pagina iniziale, è innanzitutto possibile visualizzare una schermata interattiva per la creazione di nuovi scenari di simulazione di distribuzione e per la gestione di tutti i parametri di input relativi, descritti nei precedenti paragrafi.

Il pulsante New Scenario consente di creare un nuovo scenario, a cui associare i file in formato csv necessari per effettuare la simulazione (pulsante CSV); lo scenario può essere salvato con un suo nome distintivo (SAVE NAME) per essere eventualmente richiamato in successivi momenti.

Tutti i dati di input possono essere anche caricati interattivamente dall'utente attraverso opportune maschere accessibili.

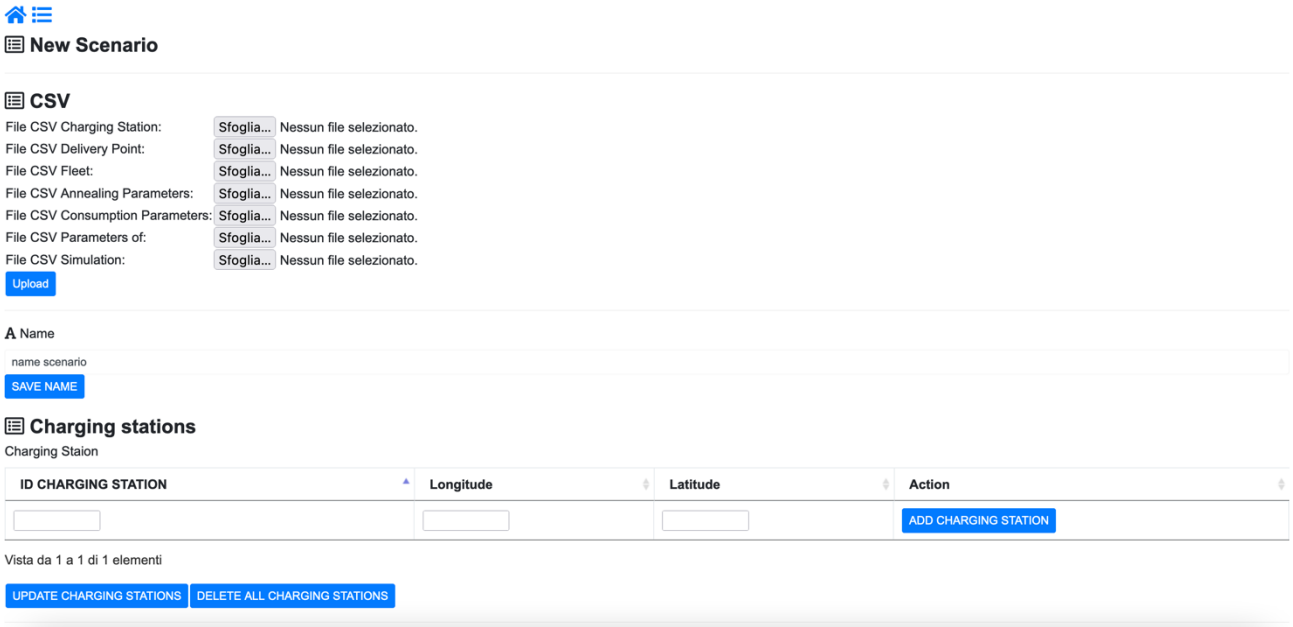


Figura 17: Pagina di inserimento e aggiornamento dati di input e parametri per lo svolgimento delle simulazioni

E' inoltre possibile visualizzare su mappa la posizione dei punti di consegna, delle colonnine di ricarica e dell'hub dell'operatore logistico. A queste entità puntuali sono associate diverse informazioni (peso delle consegne e time-windows associati ai punti di consegna, potenza della stazione di ricarica, composizione della flotta disponibile presso l'hub, ecc.) visualizzabili a video attraverso passaggio del mouse.

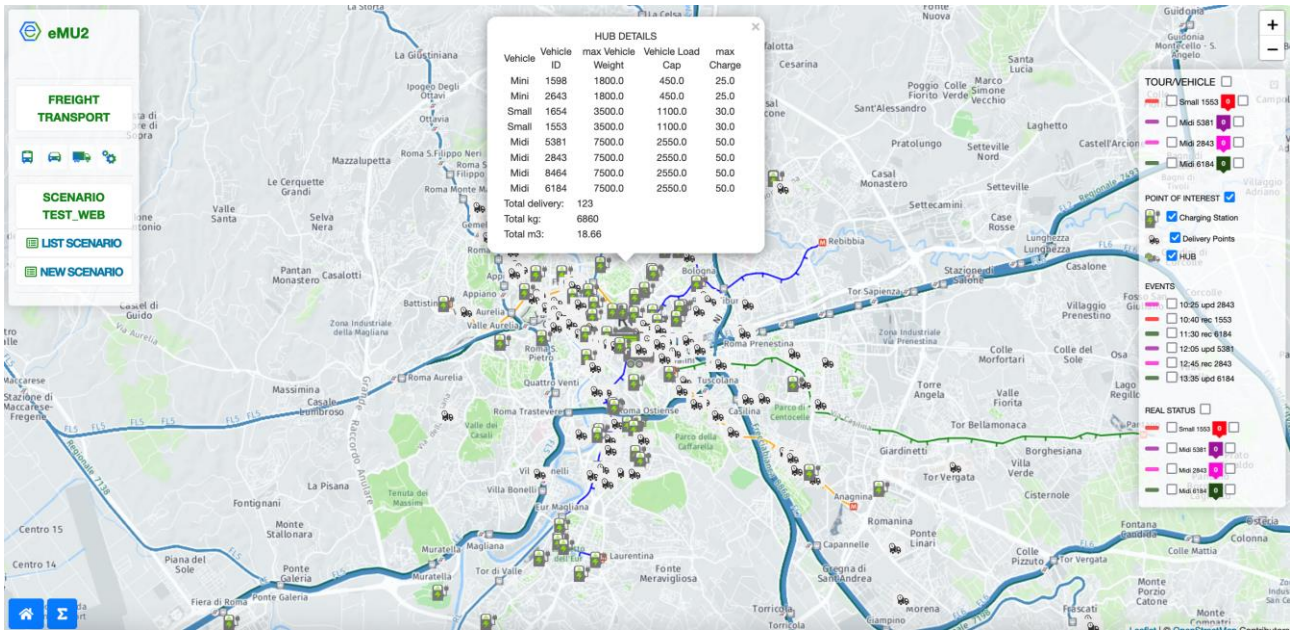


Figura 18: Visualizzazione dell'insieme dei punti di ricarica e dei punti di consegna e delle informazioni ad essi associate

Tutti gli scenari salvati sono archiviati in una banca dati da cui possono essere richiamati con molta facilità sia per delle semplici visualizzazioni che per delle modifiche utili a nuove elaborazioni, come mostrato in Figura 19.

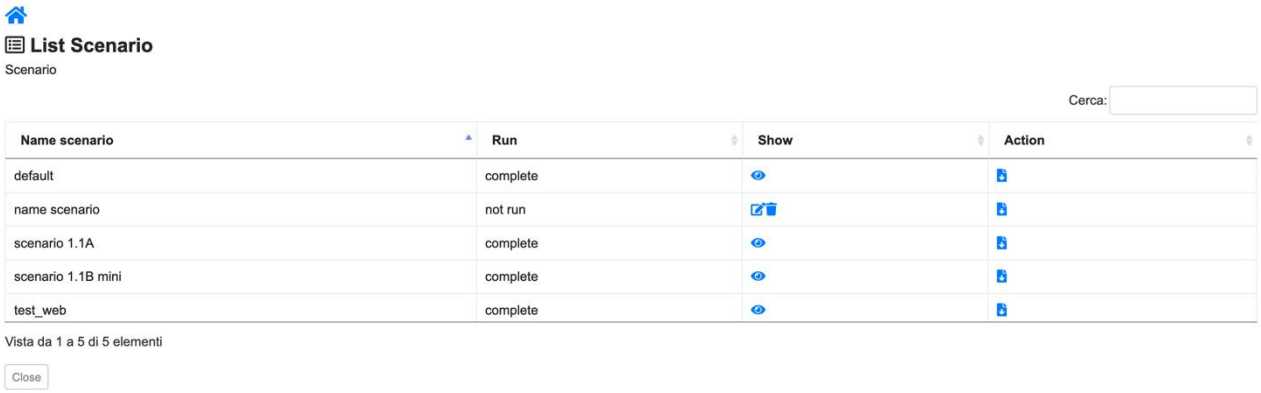


Figura 19: Pagina di gestione delle simulazioni già eseguite e in fase di esecuzione

Una volta eseguita la simulazione, il sistema consente la visualizzazione dei risultati su mappe georeferenziate. I file CSV di output descritti nei paragrafi precedenti rappresentano la base per la creazione di diversi layers grafici, uno per ciascun giro di consegna suggerito. Si rimanda alla Figura 14 per la visualizzazione dei giri di consegna dopo la fase di ottimizzazione e simulazione.

Per ogni percorso programmato è possibile attivare o disattivare la visualizzazione delle soste (consegne o ricariche). Inoltre, a fine giornata, è possibile visualizzare in modo indipendente ogni recovery e update e infine il percorso realmente effettuato dai mezzi, con o senza i punti di consegna. Altri esempi delle capacità di visualizzazione dei percorsi e delle fermate sono rappresentati in precedenza dalla Figura 15 e dalla Figura 16.

Come mostrato nell'esempio di Figura 20, è inoltre possibile visualizzare le tabelle riassuntive, un esempio delle quali è riportato nella Tabella 16 con i valori cumulati dei più importanti risultati delle simulazioni e del monitoraggio (consumi totali, km totali percorsi e Tonnellate-km).

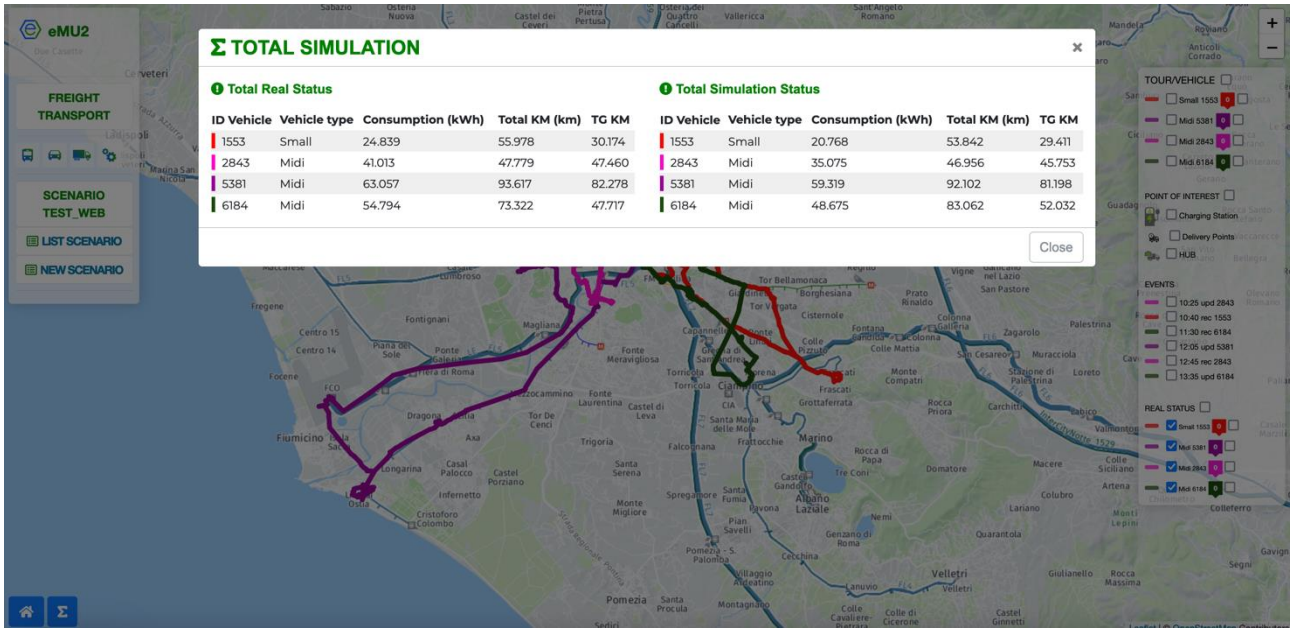


Figura 20: Esempio di una tabella riassuntiva con i dati di consumo, percorrenza e tonnellate-km relativi alla simulazione iniziale e al percorso reale

Tabella 16: Struttura dei file di sintesi SimTable.csv e RealTable.csv

Vehicle	ConsTot	Totkm	Tgkm	VehType
2843	57.884	86.510	88.390	Midi
5381	47.508	65.824	71.924	Midi
6184	61.792	82.277	86.477	Midi
8464	23.282	39.594	18.350	Midi

4 Bibliografia

[1] A. Alessandrini, F. Filippi e F. Ortenzi, «Consumption calculation of vehicles using OBD data,» in *20th International Emission Inventory Conference*, August 2012.

[2] F. Ortenzi e M. Costagliola, «A New Method to Calculate Instantaneous Vehicle Emissions using OBD Data,» *SAE Technical Paper*, vol. 2010, n. 01, p. 1289, 2010.

[3] CSS Electronics, «CAN Bus Explained, a simple intro,» 2021. [Online]. Available: https://www.csselectronics.com/pages/can-bus-simple-intro-tutorial.

[4] CSS Electronics, «OBD-II Explained, a simple Intro,» 2021. [Online]. Available: https://www.csselectronics.com/pages/obd2-explained-simple-intro.

[5] A. K. M. Kadhar e G. Arnand, Data Science with Raspberry PI, Springer, 2021.

[6] S. C. Williams, «Analysis of the SSH Key Exchange Protocol,» in *13th IMA International Conference, IMACC 2011*, Oxford, UK, December 12-15, 2011.

[7] D. L. Applegate, R. M. Bixby, V. Chavatal e W. J. Cook, The Traveling Salesman Problem, 2006.

[8] L. Perron e V. Furnon, «OR-Tools,» [Online]. Available: https://developers.google.com/optimization/.

[9] P. Toth e D. Vigo, The Vehicle Routing Problem, SIAM, Monographs on Discrete Mathematics and its Applications, 2002.

[10] OpenStreetMap contributors , «Planet dump retrieved from https://planet.osm.org,» 2022. [Online]. Available: https://www.openstreetmap.org.

[11] P. J. M. van Laarhoven e E. H. L. Aarts, Simulated annealing: theory and applications, Boston, Norwell, MA, USA: D. Reidel, 1987.

5 Allegati

5.1 Procedura di installazione del Sistema Operativo Raspbian su Raspberry PI

Il Sistema Operativo di riferimento per Raspberry PI, è denominato Raspbian (Raspberry PI OS), ed è basato sulla distribuzione Debian di Linux.

Il Sistema Operativo Raspbian è fornito direttamente dalla Raspberry PI Foundation, e va installato su una SD Card che dovrà poi essere inserita sul Raspberry PI, nel suo apposito alloggiamento.

Al fine di facilitare il processo di installazione, è possibile utilizzare il software denominato NOOBS, anch'esso fornito dalla Raspberry PI Foundation. Il primo passo consiste nell'eseguire il download del software NOOBS su un PC o su un Notebook su cui sia disponibile un lettore di SD Card.

Quindi occorre formattare la SD Card, estrarre l'archivio scaricato tramite NOOBS, ed eseguire la copia dei file sulla SD Card.

Estrarre quindi la SD Card dal lettore del PC ed inserirla nell'apposito alloggiamento sul Raspberry PI.

Avviando adesso il Rapsberry PI, si attiverà una procedura guidata che eseguirà l'installazone del Sistema Operativo Raspbian sul Raspberry PI (Figura 21).

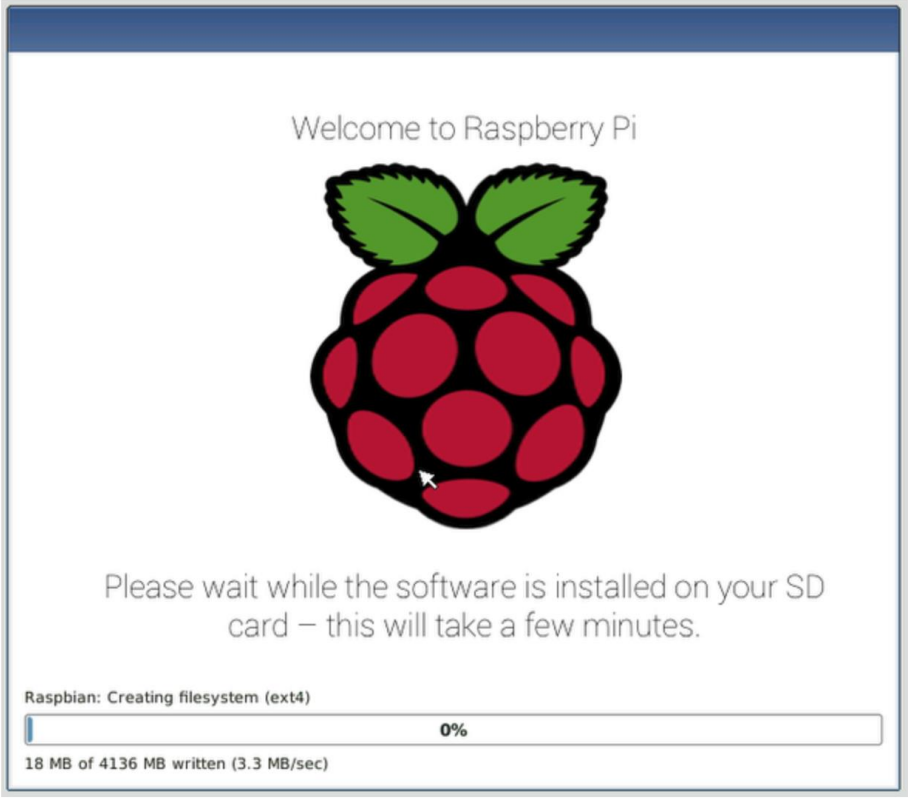


Figura 21: Logo del Raspberry PI al termine dell'installazione del Sistema Operativo Raspbian

La descrizione dell'intero processo è disponibile al seguente link:

Projects | Computer coding for kids and teens | Raspberry Pi

5.2 Installazione del Driver del dispositivo CAN-USB sul Raspberry PI

Per fare interagire il Rasperry PI con il dispositivo CAN-USB occorre innanzitutto individuare un dispositivo che renda disponibile una libreria software di driver che sia compatibile con il Raspberry PI.

Un dispositivo con queste caratteristiche, che possiede anche una libreria software di API per operare con il firmware del dispositivo, è quello messo a disposizione dalla Peak System[5], denominato PCAN-USB.

Per installare il driver del dispositivo PCAN-USB sul Raspberry PI, occorre innanzitutto eseguire il download del sotware peak-linux-driver, disponibile al seguente link:

Linux PCAN Driver: Overview (peak-system.com)

Diversamente dal caso di utilizzo di un sistema operativo Windows-like, il driver non è fornito come un pacchetto autoinstallante, ma come un archivio tar.gz da compilare sulla macchina di destinazione.

Dopo il download occorre operare da terminale, ed eseguire il seguente comando per decomprimere e estrarre l'archivio: > tar –xzvf peak-linux-driver-8.12.0.tar.gz

Spostarsi quindi nella directory appena creata con il comando : > cd peak-linux-driver-8.12.0

E successivamente eseguire il comando : > make clean per ripulire il sistema da eventuali precedenti installazioni.

Quindi avviare la compilazione dei sorgenti del driver con il comando : > make e l'installazione vera e propria del driver con il comando: > make install

È buona pratica a questo punto riavviare il Raspberry PI.

Per verificare quindi la correttezza delle operazioni eseguite, occorre collegare il dispositivo PCAN alla porta USB del Raspberry PI, ed eseguire alcuni comandi di controllo:

[5] https://www.peak-system.com

- Il comando > dmesg | grep pcan (Figura 23) elenca tutte le interfacce PCAN, ciascuna con le proprie caratteristiche, rilevate dal sistema al suo avvio. Nella figura seguente è riportato un esempio di output del comando:

```
$ dmesg | grep pcan
[24612.510888] pcan: Release_YYYYMMMDD_n (le)
[24612.510894] pcan: driver config [mod] [isa] [pci] [pec] [dng] [par] [usb] [pcc]
[24612.511057] pcan: uCAN PCI device sub-system ID 14h (4 channels)
[24612.511125] pcan 0000:01:00.0: irq 48 for MSI/MSI-X
[24612.511140] pcan: uCAN PCB v4h FPGA v1.0.5 (design 3)
[24612.511146] pcan: pci uCAN device minor 0 found
[24612.511148] pcan: pci uCAN device minor 1 found
[24612.511150] pcan: pci uCAN device minor 2 found
[24612.511153] pcan: pci uCAN device minor 3 found
[24612.516206] pcan: pci device minor 4 found
[24612.516230] pcan: pci device minor 5 found
[24612.516258] pcan: pci device minor 6 found
[24612.516280] pcan: pci device minor 7 found
[24612.516335] pcan: isa SJA1000 device minor 8 expected (io=0x0300,irq=10)
[24612.516369] pcan: isa SJA1000 device minor 9 expected (io=0x0320,irq=5)
[24612.516999] pcan: new high speed usb adapter with 2 CAN controller(s) detected
[24612.517237] pcan: PCAN-USB Pro FD (01h PCB01h) fw v2.1.0
[24612.517244] pcan: usb hardware revision = 1
[24612.517605] pcan: PCAN-USB Pro FD channel 1 device number=30
[24612.517729] pcan: usb device minor 0 found
[24612.517732] pcan: usb hardware revision = 1
[24612.518231] pcan: PCAN-USB Pro FD channel 2 device number=31
[24612.518354] pcan: usb device minor 1 found
[24612.522469] pcan: new usb adapter with 1 CAN controller(s) detected
[24612.522491] pcan: usb hardware revision = 28
[24612.579450] pcan: PCAN-USB channel device number=161
[24612.579453] pcan: usb device minor 2 found
[24612.579487] usbcore: registered new interface driver pcan
[24612.586265] pcan: major 249.
```

Figura 22: Output del comando dmesg

- Il comando > lsusb elenca le caratteristiche generali delle interfacce USB rilevate dal sistema. In questo modo è possibile rilevare a quale porta USB è collegato il PCAN, e se il dispositivo è rilevato correttamente.
- Un altro comando utile è quello che permette di visualizzare il contenuto del file di sistema "/proc/pcan". Questo file contiene diverse informazioni interessanti, quali ad esempio la versione del driver installato e l'elenco di tutte le interfacce PCAN rilevate dal sistema. Il comando in questione è il seguente: > cat /proc/pcan
- Il comando > lspcan infine, consente di elencare tutte le interfacce PCAN e i canali CAN sul sistema.

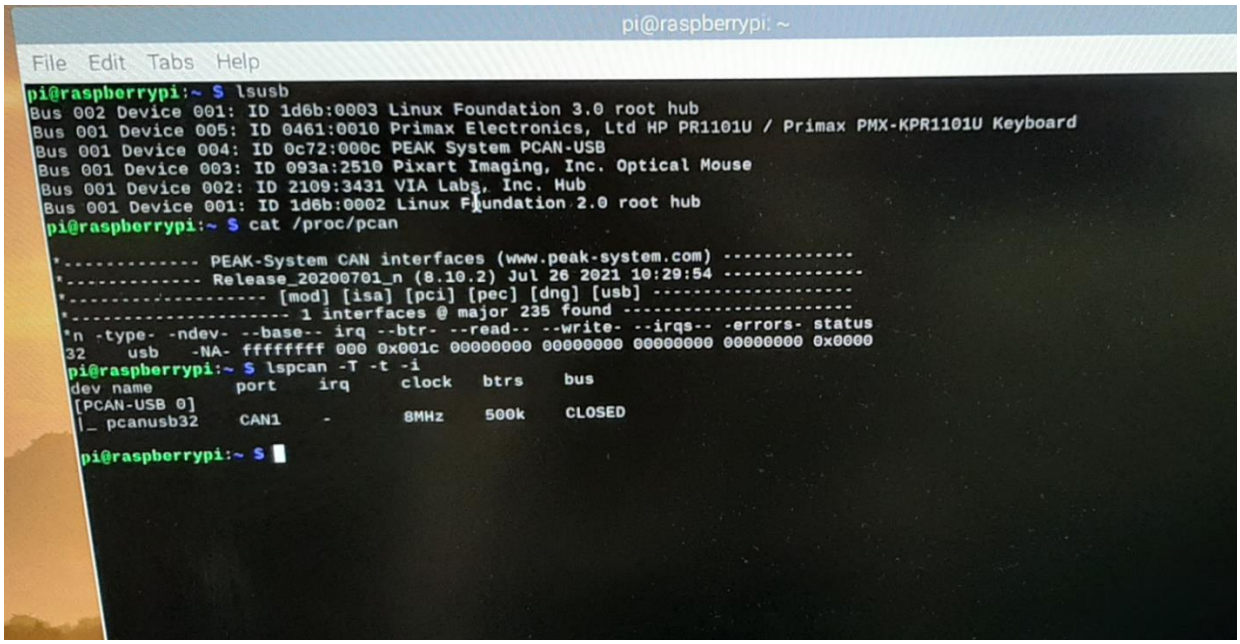


Figura 23: I comandi di verifica del driver PCAN USB

La Figura 24 mostra l'output dei comandi di controllo ottenuto nel nostro caso, dopo l'installazione del driver PCAN sul Raspberry PI.

5.3 Il Software di acquisizione dati dal veicolo

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <asm/types.h>
#include <math.h>
#include <time.h>

#ifndef NO_RT
#include <sys/mman.h>

#ifdef RTAI
#include <rtai_lxrt.h>
#endif

// PCAN-Basic device used to read on (RT version doesn't handle USB devices)
#define PCAN_DEVICE		PCAN_PCIBUS1
#else

// PCAN-Basic device used to read on
#define PCAN_DEVICE		PCAN_USBBUS1
#endif
```

```
// definizioni inserite da Gtom
#define SOC 1371                        // ID= 55B
#define M_AMP_RPM_KW 474                        // ID= 1DA
#define BATT_A_V_KW 475                 // ID= 1DB
#define AV_BATT_KW_MAX 476                      // ID= 1DC
#define VLEFT_RIGHT_VV 644                      // ID= 284
#define QC_VOLTAGE 896                  // ID= 380
#define AC_HEAT_PWR 1359                // ID= 54F
#define CHARGER_RIM_MIN 1465            // ID= 5B9
#define KWH 1468                        // ID= 5BC
#define CHARGE_A_V 1471                 // ID= 5BF
#define PI 3.14159265

#include "PCANBasic.h"

// *** Funzione per trasformare un numero decimale in notazione binaria
void DecToBin(int dec, int bitvett[]) {

        int i;
        for (i=7;i>=0;i--) {
                if ((dec-(int)pow(2,i)) < 0) bitvett[7-i]= 0;
                else {
                        bitvett[7-i]= 1;
                        dec= dec-(int)pow(2,i);
                }
        }
}




//***
float getSOC(int dec0, int dec1) {

        int bitvett0[8], bitvett1[8];
        float soc_value=0.0;
        int i;

        DecToBin(dec0, bitvett0);
        DecToBin(dec1, bitvett1);

        for (i=9;i>=0;i--)
                if (i>1) soc_value= soc_value + bitvett0[9-i] * pow(2,i); // la posizione nel vettore e' 9-i, il peso e' i
                else soc_value= soc_value + bitvett1[1-i] * pow(2,i);
        return soc_value/1024*100;
}
//***
float getHeater(int msgdata) {

        int bitvett[8];
        float heater_value=0.0;
        int i;

        DecToBin(msgdata, bitvett);
```

```
	for (i=0;i<6;i++)
		heater_value= heater_value + bitvett[7-i] * pow(2,i);

	return heater_value*250/1000;
}
//***
float getAmp(int dec2, int dec3) {

	int bitvett2[8], bitvett3[8];
	float amp_value=0.0;
	int ctrl_value;
	int i; int segno;

	DecToBin(dec2, bitvett2);
	DecToBin(dec3, bitvett3);
	segno= bitvett2[5];

	for (i=9;i>=0;i--)
		if (i>7) amp_value= amp_value + bitvett2[i-8] * pow(2,i);
		else amp_value= amp_value + bitvett3[7-i] * pow(2,i);
	ctrl_value= (int) amp_value;
	amp_value= (amp_value - segno*1024 + segno *1)/2;
	if (ctrl_value== 0) amp_value= 0.0;
	else	if ((ctrl_value== 1) && (segno== -1)) amp_value= 0.0;
			else if ((ctrl_value== 2023) && (segno == 0)) amp_value= 0.0;
	return amp_value;
}




//***
float getRpm(int dec4, int dec5) {

	int bitvett4[8], bitvett5[8];
	float out_value=0.0;
	int ctrl_value;
	int i; int segno;

	DecToBin(dec4, bitvett4);
	DecToBin(dec5, bitvett5);
	segno= bitvett4[0];

	for (i=13;i>=0;i--)
		if (i>7) out_value= out_value + bitvett4[15-i] * pow(2,i);
		else out_value= out_value + bitvett5[7-i] * pow(2,i);
	ctrl_value= (int) out_value;
	out_value= (out_value - segno*32768 + segno *1)/2;
	if (ctrl_value== 0) out_value= 0.0;
	else	if ((ctrl_value== 1) && (segno== -1)) out_value= 0.0;
			else if ((ctrl_value== 32767) && (segno == 0)) out_value= 0.0;
	return out_value;
}
//***
float getBattA(int dec0, int dec1) {
```

```
	int bitvett0[8], bitvett1[8];
	float out_value=0.0;
	int ctrl_value;
	int i; int segno;

	DecToBin(dec0, bitvett0);
	DecToBin(dec1, bitvett1);
	segno= bitvett0[0];

	for (i=8;i>=0;i--)
		if (i>2) out_value= out_value + bitvett0[10-i] * pow(2,i);
		else out_value= out_value + bitvett1[2-i] * pow(2,i);
	ctrl_value= (int) out_value;
	out_value= (out_value - segno*1024 + segno *1)/2;
	if (ctrl_value== 0) out_value= 0.0;
	else if ((ctrl_value== 1) && (segno== -1)) out_value= 0.5;
		else if ((ctrl_value== 1023) && (segno == 0)) out_value= -0.5;

	return out_value;
}


//***
float getBattV(int dec2, int dec3) {

	int bitvett2[8], bitvett3[8];
	float out_value=0.0;
	int ctrl_value;
	int i;

	DecToBin(dec2, bitvett2);
	DecToBin(dec3, bitvett3);

	for (i=9;i>=0;i--)
		if (i>1) out_value= out_value + bitvett2[9-i] * pow(2,i);
		else out_value= out_value + bitvett3[1-i] * pow(2,i);
	ctrl_value= (int) out_value;
	if (ctrl_value== 1023) out_value= 0.0;
	else	out_value= out_value/2;
	return out_value;
}
//***
float getMaxBattery(int dec1, int dec2) {

	int bitvett1[8], bitvett2[8];
	float out_value=0.0;
	int ctrl_value;
	int i;

	DecToBin(dec1, bitvett1);
	DecToBin(dec2, bitvett2);

	for (i=11;i>=0;i--)
		if (i>3) out_value= out_value + bitvett1[11-i] * pow(2,i);
		else out_value= out_value + bitvett2[3-i] * pow(2,i);
	return out_value;
```

```
}
//***
float getVleft(int dec0, int dec1) {

        int bitvett0[8], bitvett1[8];
        float out_value=0.0;
        int i; int segno;

        DecToBin(dec0, bitvett0);
        DecToBin(dec1, bitvett1);
        segno= bitvett0[0];

        for (i=14;i>=0;i--)
                if (i>7) out_value= out_value + bitvett0[15-i] * pow(2,i);
                else out_value= out_value + bitvett1[7-i] * pow(2,i);
        out_value= (out_value - segno*32768 + segno *1)/200;
        return out_value;
}



//***
float getVright(int dec2, int dec3) {

        int bitvett2[8], bitvett3[8];
        float out_value=0.0;
        int i; int segno;

        DecToBin(dec2, bitvett2);
        DecToBin(dec3, bitvett3);
        segno= bitvett2[0];

        for (i=14;i>=0;i--)
                if (i>7) out_value= out_value + bitvett2[15-i] * pow(2,i);
                else out_value= out_value + bitvett3[7-i] * pow(2,i);
        out_value= (out_value - segno*32768 + segno *1)/200;
        return out_value;
}
//***
float getVehicle(int dec4, int dec5) {

        int bitvett4[8], bitvett5[8];
        float out_value=0.0;
        int i; int segno;

        DecToBin(dec4, bitvett4);
        DecToBin(dec5, bitvett5);
        segno= bitvett4[0];
        for (i=14;i>=0;i--)
                if (i>7) out_value= out_value + bitvett4[15-i] * pow(2,i);
                else out_value= out_value + bitvett5[7-i] * pow(2,i);
        out_value= (out_value - segno*32768 + segno *1)/100;
        return out_value;
}
//***
float getRemain(int dec0, int dec1) {
```

```
	int bitvett0[8], bitvett1[8];
	float out_value=0.0;
	int i;

	DecToBin(dec0, bitvett0);
	DecToBin(dec1, bitvett1);

	for (i=10;i>=0;i--)
		if (i>7) out_value= out_value + bitvett0[15-i] * pow(2,i);
		else out_value= out_value + bitvett1[7-i] * pow(2,i);
	return out_value;
}

//***
float getKwh(int dec0, int dec1) {

	int bitvett0[8], bitvett1[8];
	float out_value=0.0;
	int i;

	DecToBin(dec0, bitvett0);
	DecToBin(dec1, bitvett1);

	for (i=9;i>=0;i--)
		if (i>1) out_value= out_value + bitvett0[9-i] * pow(2,i);
		else out_value= out_value + bitvett1[1-i] * pow(2,i);
	return out_value* 0.085;
}




//***
float getDec(int dec) {

	int bitvett[8];
	float out_value= 0.0;
	int i;

	DecToBin(dec,bitvett);
	for (i=7;i>=0;i--) out_value= out_value + bitvett[i] * pow(2,i);
	return out_value;
}

static void signal_handler(int s)
{
	printf("Interrupted by SIG%u!\n", s);
}
////////////////////////////////////////////////////////////////////////////////////////////////
/// <summary>	Main entry-point for this application. </summary>
///
/// <remarks>	 </remarks>
///
/// <param name="argc">	The argc. </param>
/// <param name="argv">	[in,out] If non-null, the argv. </param>
///
```

```
/// <returns>           . </returns>
////////////////////////////////////////////////////////////////////////////////////////////////////
int main(int argc, char* argv[])
{
        TPCANMsg Message;
        TPCANStatus Status;
        unsigned int pcan_device = PCAN_DEVICE;
        int i=0; int rowid;
        int rows= 1000;                                 // legge rows righe e poi esce
        FILE *fp;                                       // file di log
        char fileout[]= "datalog.txt";
        int msgID, msgLen, msgData0, msgData1, msgData2;
        int msgData3, msgData4, msgData5, msgData6, msgData7;
        int invv, av_battery, qcVoltage, qcComm;
        float socval, acpwr, heater_pwr, motorAmp, motorRpm, motorKw;
        float batt_A, batt_V, max_battery, battKw;
        float vleft, vright, vvehicle, chargeRem, kwh, chargeCurrent, chargeVoltage;
        char day[100], mytime[100];

        time_t current_time;   // = time(NULL);
        struct tm *tmptr;     // = localtime(&current_time);

#ifndef NO_RT
        mlockall(MCL_CURRENT | MCL_FUTURE);

#ifdef RTAI
        // Initialize LXRT
        RT_TASK *mainr = rt_task_init_schmod(nam2num("MAINR"), 0, 0, 0,
                                                SCHED_FIFO, 0xF);
        if (!mainr) {
                printf("pcanread(%xh): unable to setup main RT task\n",
                        PCAN_DEVICE);
                return -1;
        }
        rt_make_hard_real_time();
#endif
#endif

        // get the device from the cmd line if provided
        if (argc > 1) {
                char *endptr;
                unsigned long tmp = strtoul(argv[1], &endptr, 0);
                if (*endptr == '\0')
                        pcan_device = tmp;
        }

        // below usleep() will be INTRuptible by user
        signal(SIGINT, signal_handler);

        Status = CAN_Initialize(pcan_device, PCAN_BAUD_500K, 0, 0, 0);
        printf("CAN_Initialize(%xh): Status=0x%x\n", pcan_device, (int)Status);
        if (Status)
                goto lbl_exit;

        printf("after goto");
        // inizializzo tutte le variabili
        socval= 0.0; invv=0.0; motorAmp= 0.0; motorRpm=0.0; motorKw=0.0;
```

```
batt_A=0.0; batt_V=0.0; battKw=0.0;
av_battery=0; max_battery=0.0;
vleft=0.0; vright= 0.0; vvehicle=0.0;
qcVoltage= 0; qcComm= 0;
acpwr=0.0; heater_pwr=0.0;
chargeRem= 0.0;
kwh= 0.0;
chargeCurrent=0.0; chargeVoltage= 0.0;


fp= fopen(fileout,"w");
while (i< rows) {
        while ((Status=CAN_Read(pcan_device, &Message, NULL)) == PCAN_ERROR_QRCVEMPTY)
                if (usleep(100))
                        break;

        if (Status != PCAN_ERROR_OK) {
                printf("CAN_Read(%xh) failure 0x%x\n", pcan_device, (int)Status);
                break;
        }

        //printf("  - R ID:%4x LEN:%1x DATA:%02x %02x %02x %02x %02x %02x %02x %02x\n",
        msgID= (int)Message.ID;
        msgLen= (int)Message.LEN;
        msgData0= (int)Message.DATA[0];
        msgData1= (int)Message.DATA[1];
        msgData2= (int)Message.DATA[2];
        msgData3= (int)Message.DATA[3];
        msgData4= (int)Message.DATA[4];
        msgData5= (int)Message.DATA[5];
        msgData6= (int)Message.DATA[6];
        msgData7= (int)Message.DATA[7];

        rowid= (int)Message.ID;
        switch (rowid) {

                case SOC:  socval= getSOC(msgData0,msgData1);
                                        break;
                case M_AMP_RPM_KW:      invv= msgData0 *2;
                                                motorAmp= getAmp(msgData2, msgData3);
                                                motorRpm= getRpm(msgData4, msgData5);
                                                motorKw= 2*PI*((motorRpm/60)*(motorAmp/1000));
                                                break;
                case BATT_A_V_KW:  batt_A= (-1) * getBattA(msgData0, msgData1);
                                                batt_V= getBattV(msgData2, msgData3);
                                                battKw= (batt_A * batt_V) /1000;
                                                break;
                case AV_BATT_KW_MAX:    av_battery= msgData0;
                                        max_battery= getMaxBattery(msgData1, msgData2);
                                        //printf("av_battery %d\n", msgData0);
                                        break;
                case VLEFT_RIGHT_VV:    vleft= getVleft(msgData0, msgData1);
                                        vright= getVright(msgData2, msgData3);
                                        vvehicle= getVehicle(msgData4, msgData5);
                                        break;
                case QC_VOLTAGE:    //qcVoltage= getDec(msgData3);
                                        qcVoltage= msgData3;
```

```
                                        qcComm= msgData4;
                                        //qcComm= getDec(msgData4);
                                        //printf("QC %d -- %d\n", msgData3,msgData4);
                                        break;

                    case AC_HEAT_PWR: acpwr= (float)msgData2*50/1000;
                                                heater_pwr= getHeater(msgData5);
                                                break;
                    case CHARGER_RIM_MIN:       chargeRem= getRemain(msgData0, msgData1);
                                                        break;
                    case KWH:           kwh= getKwh(msgData0, msgData1);
                                        break;
                    case CHARGE_A_V:    chargeCurrent= (float)(msgData6 / 8);
                                                chargeVoltage= (float) (msgData3 * 3);
                                                break;
                    default:                    //socval= 0.0; invv=0.0; motorAmp= 0.0; motorRpm=0.0; motorKw=0.0;
                                                //batt_A=0.0; batt_V=0.0; battKw=0.0;
                                                //av_battery=0.0; max_battery=0.0;
                                                //vleft=0.0; vright= 0.0; vvehicle=0.0;
                                                //qcVoltage= 0.0; qcComm= 0.0;
                                                //acpwr=0.0; heater_pwr=0.0;
                                                //chargeRem= 0.0;
                                                //kwh= 0.0;
                                                //chargeCurrent=0.0; chargeVoltage= 0.0;
                                                break;
                }

#ifdef XENOMAI
                // force flush of printf buffers
                rt_print_flush_buffers();
#endif
        i++;
        //}
  current_time = time(NULL);
  tmptr = localtime(&current_time);
  strftime(day, sizeof(day), "%Y%m%d", tmptr);
  strftime(mytime, sizeof(mytime), "%H%M%S", tmptr);

fprintf(fp, "%s;%s;%f;%d;%f;%f;%f;%f;%f;%d;%f;%f;%f;%f;%f;%d;%d;%f;%f;%f;%f;%f;%f\n", day,mytime,
        socval, invv, motorAmp, motorRpm, motorKw, batt_A, batt_V, av_battery, max_battery, battKw, vleft, vright, vvehicle,
                                        qcVoltage, qcComm, acpwr, heater_pwr, chargeRem, kwh, chargeCurrent, chargeVoltage);

}
fclose(fp);
CAN_Uninitialize(pcan_device);

lbl_exit:
#ifdef XENOMAI
#elif defined(RTAI)
        rt_make_soft_real_time();
        rt_task_delete(mainr);
#endif

return 0;
```

5.4 Lettura dati dalla porta seriale USB

Al fine di automatizzare il processo di lettura e codifica dei messaggi inviati dal ricevitore GPS, è stato messo a punto uno script (Figura 24) in linguaggio Python[6], che acquisisce un numero definito a priori di tali messaggi, li interpreta e salva su un file di uscita tutte le informazioni di interesse.

Basterà poi mandare in esecuzione lo script ad intervalli di tempo regolari, per ottenere le informazioni della posizione correlate agli istanti temporali.

```
import serial
import string
import time

from subprocess import call



PORT = "/dev/ttyACM0"

s= serial.Serial(PORT)

s.flush()
line= s.readline()
print(line)
counter=1
while counter < 20 :
        line= s.readline()
        print(line)
        if (line[0:6]== "$GNGGA") :
                fields= line.split(",")
                day=time.strftime("%Y-%m-%d")
                timeofday= time.strftime("%H:%M:%S")
                gpstime= fields[1]
                gpslat= fields[2]
                gpslon= fields[4]
        counter+=1
#s.close()


latgrad=gpslat[0:2]
latprimi=gpslat[2:-1]
longrad= gpslon[0:3]
lonprimi= gpslon[3:-1]
#print(latgrad+ " " + latprimi+ " "+longrad + "  " + lonprimi + " --")
latdec= float(latgrad)+ float(latprimi)/60
londec= float(longrad)+ float(lonprimi)/60
#print("day= " + day + " time= " + timeofday + " gpstime= " + gpstime + " latitudine= " + str(latdec) +  " -- longitudine= " + str(londec))
vID= 1;  #codice numerico del veicolo
hourofday=time.strftime("%H")
minuteofhour=time.strftime("%M")

minuteFromMidnight= int(hourofday)*60+int(minuteofhour)

lineout= str(vID)+";"+day+" "+timeofday+";"+str(minuteFromMidnight)+";"+str(latdec)+";"+str(londec)
with open('./PCAN/gpsData.csv','w') as fw:
        fw.write(lineout)

~
~
```

Figura 24: Lo script "readSerial.py"

Lo script legge i messaggi in ingresso alla porta seriale di riferimento per il ricevitore GPS, e per ciascuna riga acquisita, legge il contenuto dell'Address field per capire il tipo di informazione che sta per essere ricevuta, e poi legge e memorizza l'informazione stessa.

5.5 Generazione di chiavi SSH

[6] Welcome to Python.org

Per generare una coppia di chiavi per un host client, occorre eseguire il comando ***ssh-keygen*** seguito dalle opzioni di configurazione desiderate. Ad esempio è possibile specificare l'algoritmo di crittografia da utilizzare con l'opzione –t, oppure il numero di bit nella chiave con l'opzione –b. Dopo aver generato la coppia di chiavi occorre propagare sul server la chiave pubblica. Questa operazione può essere eseguita utilizzando il comando ***ssh-copy-id*** <nome_chiave_pubblica> <nome_server>

La Figura 25 mostra il processo di creazione di una coppia di chiavi pubblica e privata sul dispositivo Raspberry PI 4. La chiave pubblica viene poi condivisa con il server tramite il comando ***ssh-copy-id***

Figura 25: Generazione di una coppia di chiavi pubblica e privata

5.6 Trasferire dati al server remoto

```
import sys
import time
from subprocess import call

#call(['sudo', 'pon','myvpn'])
#time.sleep(25)

from ftplib import FTP

ftp= FTP('192.168.132.184')
ftp.login(user=' *** ',passwd=' ********')
ftp.cwd('Apic-Data')

nomefile= sys.argv[1]
call(['cp','./PCAN/'+nomefile,'./'])

def placeFile(filename):

    datafilename= time.strftime("%Y%m%d%H%M%S_")+filename
    ftp.storbinary('STOR '+datafilename, open(filename, 'rb'))
    ftp.quit()

placeFile(nomefile)

dataname= time.strftime("%Y%m%d%H%M%S_")+nomefile
call(['mv',nomefile,'./SentData/'+dataname])
if nomefile == 'gpsData.csv':
        call(['scp','./SentData/'+dataname,'giuseppe@apic.casaccia.enea.it:~/gpsData/'+dataname])
#call(['sudo', 'poff','myvpn'])
~
```

Figura 26: Lo script di trasferimento dati

Lo script *uploadftp.*py (Figura 26) si collega al server remoto FTP per inviare il file indicato sulla linea di comando al momento dell'avvio dello script, ma al termine del caricamento, invia lo stesso file anche al server di archiviazione in modalità SSH sicura. A tal fine utilizza il comando ***scp*** per copiare i file tramite connessione SSH, attraverso la chiamata di sistema ***call.***
Avendo già definito la coppia di chiavi pubblica e privata, la connessione e la contestuale copia dei files, avviene in modalità del tutto automatica, e non occorre specificare alcuna password di accesso.

Da notare all'inizio e alla fine dello script, la chiamata di sistema ***call***, utilizzata per aprire o chiudere una eventuale connessione VPN. La descrizione dettagliata di questa istruzione viene rimandata al prossimo paragrafo.

5.7 Connessione alla rete VPN

Per i nostri scopi, è stato predisposto uno script di connessione alla rete VPN dell'ENEA, che può essere personalizzato e generalizzato per il collegamento ad una qualsiasi rete VPN aziendale.
Lo script viene eseguito prima di effettuare la trasmissione dei dati, per proiettare il Raspberry PI all'interno della rete locale sulla quale si trova il Server di destinazione.

Da linea di comando su terminale, per avviare lo script, basterebbe digitare il seguente comando di sistema: ***sudo myvpn pon.*** Analogamete, per bloccare l'esecuzione, basterebbe digitare il seguente comando di sistema: ***sudo myvpn poff***

Ma dovendo automatizzare il processo, è necessario annegare il comando all'interno di uno script Python, utilizzando l'istruzione Python ***call*** che esegue una chiamata di sistema. Per questo all'interno dello script di trasferimento dei dati *"uploadftp.py"* (riportato in Fig.21), sono state inserite due istruzioni ***call***, una all'inizio per avviare la connessione VPN, ed una alla fine per terminarla.
Naturalmente, a seconda delle circostanze, le righe in cui compare la call possono essere commentate, se non occorre eseguire la connessione alla rete VPN.
La Figura 27 mostra lo script di connessione alla rete VPN.

```
pi@raspberrypi:/etc/ppp $ sudo more ./peers/myvpn
pty "pptp vpnserver.casaccia.enea.it --nolaunchpppd --debug"
name giuseppe.tomasino
password ********
remotename PPTP
require-mppe-128
require-mschap-v2
refuse-eap
refuse-pap
refuse-chap
refuse-mschap
noauth
debug
persist
maxfail 0
defaultroute
replacedefaultroute
usepeerdns
```

Figura 27: Lo script di connessione alla rete VPN

5.8 Funzionamento del sensore magnetico

Il sensore magnetico M-GAGE DX80 è in grado di rilevare le variazioni del campo magnetico terrestre nelle sue vicinanze, causate dallo spostamento di una grande massa ferrosa.
Il dispositivo è alimentato con una batteria al Litio integrata al suo interno, la cui durata, in condizioni di funzionamento regolari, è garantita per diversi anni. In questo stato il sensore invia con regolarità dei segnali radio ad un ricevitore (il controller) a cui è associato, e la gestione completa del sensore viene effettuata da remoto attraverso il controller.

Per mantenere lo stato di carica della batteria però, il dispositivo viene distribuito inizialmente, in modalità cosiddetta "*dormiente*" (***deep-sleep mode***), e deve essere attivato prima di potere essere utilizzato sul campo. Nello stato "dormiente", il sensore non trasmette alcun segnale radio e quindi non può essere rilevato dal controller. Per portarlo nello stato di funzionamento regolare, occorre quindi procedere al "risveglio" del sensore. A tal fine viene utilizzato un dispositivo attivatore, che poggiato sulla zona sensibile del sensore, lo stimola con un fascio di luce infrarossa. In seguito a tale stimolazione, il sensore attiva la modalità di funzionamento "regolare", e comincia a trasmettere segnali radio per cercare nelle vicinanze un controller a cui associarsi. Una volta effettuata l'associazione, il sensore è accessibile dal controller, e può anche essere regolato in modo da essere in grado di rilevare soltanto il movimento di una certa classe di veicoli, e non di tutti. Il sensore infatti, è in grado di rilevare la variazione del campo magnetico (locale) nelle sue vicinanze, dovuta allo spostamento di un veicolo. Se tale variazione, misurata dal sensore, supera una certa soglia preimpostata, allora il sensore segnalerà un cambiamento di stato, corrispondente alla presenza di un veicolo nelle sue vicinanze.

Durante l'installazione allora, devono essere eseguite due operazioni di base, la "misura" del campo magnetico locale, che servirà da valore di riferimento (baseline), e l'impostazione della soglia di allarme, in modo che il sensore segnali una variazione del campo magnetico di riferimento oltre la soglia preimpostata.

Queste due operazioni vengono effettuate utilizzando il software in dotazione, che è in grado di accedere in modalità read/write ai registri del dispositivo.

Il sensore M-GAGE DX80 è infatti dotato di 16 registri "Modbus" per contenere il valore istantaneo di tutti i suoi parametri di riferimento (Figura 28). Anche il Controller/Gateway possiede 16 registri, e siccome esso è sempre considerato come il dispositivo del Nodo 0, i registri individuati con i numeri identificativi da 1 a 16 sono quelli del Gateway. I registri invece dei sensori connessi al Gateway avranno numeri identificativi da 17 a 32 (per il sensore del Nodo 1), da 33 a 48 per il sensore del Nodo 3, e così via.

Modbus Register Table (M-GAGE)

I/O #	Modbus Holding Register		I/O Type
	Gateway	Any Node	
1	1	1 + (Node# × 16)	M-GAGE
		...	
7	7	7 + (Node# × 16)	Reserved
8	8	8 + (Node# × 16)	Device Message
		...	
13	13	13 + (Node# × 16)	Configuration Message
14	14	14 + (Node# × 16)	Baseline Command
15	15	15 + (Node# × 16)	Control Message
16	16	16 + (Node# × 16)	Reserved

Figura 28: Il Modbus Register del M-GAGE DX80

Per ciascun sensore (dal Nodo 1 in poi), il registro n. 1 (I/O #1) contiene il valore della deviazione tra il valore attuale del campo magnetico locale, ed il valore di riferimento (baseline) memorizzato nel registro.
Il registro n. 13 consente di inviare al sensore dei messaggi di configurazione, il registro n.14 è dedicato ai comandi per settare il valore di riferimento del campo magnetico (baseline).

5.9 Il codice in formato Python di gestione dei codici di programmazione e recupero

```
# This is the main file to handle input and outputs of the Optimization and
Recovery software handle by the
# the University La Sapienza
import shutil
from pathlib import Path
import pandas as pd
import osmnx as ox
import numpy as np
import math
import subprocess
import os
import networkx as nx
import random
import csv
## RICCARDO INIZIO
from json import dumps
import json
## RICCARDO FINE
def optim in():

    # Compute all input files for optimization
    # Launch the matlab executable 'simulation_run'

    # Open existing files (delivery points and charging points need to be
computed every time a new graph
    # is extracted since IDs are linked to nodes and links and there is a high
risk that a small change in
    # the graph brakes the link between charging/deliveri points IDs and
nodes/links.
    # charging and delivery points IDs are nodes IDs!
```

```
    #
    # Change here all required paths
    basedir = os.getcwd()
    inputdir = Path(basedir, 'exe/test_data/original')
    saveindir = Path(basedir, 'exe/test_data')
    file_graphml = Path(basedir, 'data/Roma__Italy_70km.graphml')
    matlabdir = Path(basedir, 'exe/Simulazione_v10')
    matlabruntime = Path('/usr/local/MATLAB/MATLAB_Runtime/v99/')


    timestep=[]
    # Parameters

    """
    mysimoutinterval = 10   # seconds. Frequency of detailsim output
    startfromosm = True     # Read from test_data or from test_data/original
    random.seed(10)         # Seed for random delay of real status
    np.random.seed(4)       # Random seed for velocity perturbation
    randthres = 0.8         # threshold to activate 10 s delay
    SOC10s = 0.01           # discharge in case of random delay (KWh)
    velreduction = 0.7      # Velocity reduction respect to construction
velocity
    socratio = 0.05          # Threshold for ratio between deltaSOC and SOC
    socabs = 0.2            # Threshold for percentage value of SOC for recovery
    deldelay = -3           # Threshold for update triggering, difference of
deliveries respect to expected
    """
    ## RICCARDO INIZIO
    parPython = pd.read_csv(Path(saveindir / 'parameters.csv'), sep=';')  #
Parameters python
    mysimoutinterval = int(parPython['mysimoutinterval']) # seconds. Frequency
of detailsim output
    startfromosm = True     # Read from test_data or from test_data/original
    random.seed(int(parPython['random_delay']))         # Seed for random delay
of real status
    np.random.seed(int(parPython['random_velocity']))       # Random seed for
velocity perturbation
    randthres =float(parPython['randthres'])         # threshold to activate 10
s delay
    SOC10s = float(parPython['SOC10s'])             # discharge in case of
random delay (KWh)
    velreduction = float(parPython['velreduction'])       # Velocity reduction
respect to construction velocity
    socratio = float(parPython['socratio'])         # Threshold for ratio
between deltaSOC and SOC
    socabs =float(parPython['socabs'])            # Threshold for percentage
value of SOC for recovery
    deldelay = int(parPython['deldelay'])           # Threshold for update
triggering, difference of deliveries respect to expected
    ## RICCARDO FINE

    # Set initial files
    if startfromosm:
        delivery = pd.read_csv(Path(saveindir /
'DeliveryPoints.csv'),delimiter=';')
        colonnine = pd.read_csv(Path(saveindir /
'ChargingStations_orig.csv'),delimiter=';')
    else:
        delivery = pd.read_csv(Path(inputdir /
'DeliveryPoints.csv'),delimiter=';')
        colonnine = pd.read_csv(Path(inputdir /
```

```
'ChargingStations.csv'),delimiter=';')

    # Graph... working on openstreetmap
    print('Opening Graph')
    grafo_ALL_orig = ox.load_graphml(file_graphml)
    grafo_ALL = ox.utils_graph.get_largest_component(grafo_ALL_orig,
strongly=True)
    grafo_ALL = ox.add_edge_speeds(grafo_ALL)
    grafo_ALL = ox.add_edge_travel_times(grafo_ALL)
    # get edges and nodes
    gdf_nodes_ALL, gdf_edges_ALL = ox.graph_to_gdfs(grafo_ALL)

    # convert colonnine ids
    colonnine['New_ID'] = colonnine.apply(lambda x: calcnewid(grafo_ALL,
                                          x['Charging Station Longitude'],
                                          x['Charging Station Latitude']),
                                          axis=1)
    colonnine['Charging Station ID'] = colonnine['New_ID']
    colonnine.drop(columns=['New_ID'], inplace=True)
    # convert delivery ids
    delivery['New_ID'] = delivery.apply(lambda x: calcnewid(grafo_ALL,
                                          x['Delivery point longitude'],
                                          x['Delivery point latitude']),
                                          axis=1)
    delivery['Delivery Point ID'] = delivery['New_ID']
    delivery.drop(columns=['New_ID'], inplace=True)

    # Create graph files consistent with optimization
    # Nodes:
    nodes = gdf_nodes_ALL[['osmid','x','y']].copy()
    nodes.reset_index(drop=True,inplace=True)
    nodes.rename(columns={'osmid': 'ID_Node', 'x': 'xcoord', 'y':
'ycoord'},inplace=True)
    # Links
    links = gdf_edges_ALL[['length','osmid','highway','maxspeed']].copy()
    links.reset_index(inplace=True)
    links.drop(columns=['key'],inplace=True)
    links.rename(columns={'u': 'StartNodes', 'v':'EndNodes','length':
'Length','osmid':'ID'},inplace=True)

    # Parameter to reduce construction velocity to a more reasonal velocity. To
be handled any choice on partial
    # modification of subsets
    print('Working on speeds')
    links['Speed (00:00-00-59)'] = links.apply(lambda x:
selectvel(x['maxspeed'],x['highway'],velreduction),axis=1)

    # Copy velocities for 00 to the other columns representing the 24 hours of
the day
    for hh in range(23):
        hhp1=hh+1
        if hhp1<10:
            hhp1='0'+str(hhp1)
        else:
            hhp1=str(hhp1)
        links['Speed ('+hhp1+':00-'+hhp1+'-59)'] = links['Speed (00:00-00-59)']

    # delete unneeded columns
    links.drop(columns=['maxspeed','highway'],inplace=True)
```

```
    # Give an ID to links
    links['ID'] = np.arange(links.shape[0])
    links['ID'] = links['ID'] + 1

    # Check if there are delivery with 0 weight. In case remove the row
    print('Checking Delivery and charging points')
    delivery['Remove'] = delivery.apply(lambda x: nopeso(x['Demand_kg']),axis=1)
    delivery.loc[0,['Remove']] = False
    delivery = delivery.drop(delivery[delivery.Remove == True].index)
    delivery.drop(columns=['Remove'],inplace=True)
    #print(delivery)

    # Check if there are colonnine with the same node of delivery, and remove
them.
    colonnine['Remove'] = colonnine.apply(lambda x:
nocsindp(delivery,x['Charging Station ID']),axis=1)
    colonnine = colonnine.drop(colonnine[colonnine.Remove == True].index)
    colonnine.drop(columns=['Remove'],inplace=True)

    # Write fields to csv files
    nodes.to_csv(Path(saveindir / 'Nodes.csv'),index=False,sep=';')
    links.to_csv(Path(saveindir / 'Links.csv'),index=False,sep=';')
    colonnine.to_csv(Path(saveindir /
'ChargingStations.csv'),index=False,sep=';')
    delivery.to_csv(Path(saveindir / 'DeliveryPoints.csv'),index=False,sep=';')

    # Read all other inputs
    Fleet = pd.read_csv(Path(saveindir / 'Fleet.csv'), sep=';',skiprows=1)  #
Electric Vehicle Fleet


    parAnn = pd.read_csv(Path(saveindir / 'annealingParameters.csv'), sep=';')
# Simulated Annealing Parameters
    parOF = pd.read_csv(Path(saveindir / 'parametersOF.csv'), sep=';')  #
Objective Function Parameters
    parCon = pd.read_csv(Path(saveindir / 'consumptionParameters.csv'), sep=';')
# Consumption Parameters
    parSim = pd.read_csv(Path(saveindir / 'Simulation.csv'), sep=';',skiprows=1)
# Simulation Parameters

    # execute Simulation
    origdir = os.getcwd()
    os.chdir(matlabdir)
    myproc = subprocess.Popen([Path(matlabdir /
'run_Simulazione.sh'),Path(matlabruntime)])
    #myproc = subprocess.Popen([Path(matlabdir /
'run_Simulation.sh'),Path(matlabruntime)])
    #myproc = subprocess.Popen([Path(matlabdir /
'run_Pianificazione.sh'),Path(matlabruntime)])
    myproc.wait()
    print('Fatto Matlab')
    shutil.copy2(Path(saveindir / 'OptOut.csv'),Path(saveindir /
'OptOut orig.csv'))
    shutil.copy2(Path(saveindir / 'planOut.mat'),Path(saveindir /
'planOut_orig.mat'))
    shutil.copy2(Path(saveindir / 'simOut.csv'),Path(saveindir /
'simOut_orig.csv'))
    os.chdir(origdir)

    # Start working on real events
    # Open simulation output
```

```
    simouttmp = pd.read_csv(Path(saveindir / 'simOut.csv'),delimiter=',')
    simout = simouttmp.iloc[:,:7]

    # The real path of each vehicle is derived from the simulated path, adding
random noise
    simtimes = simout['SimTime'][simout['Vehicle'] ==
simout['Vehicle'].values[0]].tolist()
    interval = parSim['timeStep'].values[0] * 60
    optout = pd.read_csv(Path(saveindir / 'OptOut.csv'), delimiter=',')
    ## RICCARDO INIZIO
    genera_file_opt_out_coord(Path(saveindir / 'OptOut.csv'),Path(saveindir /
'Nodes.csv'),Path(saveindir / 'coord_OptOut.json'))
    ## RICCARDO FINE
    mysimout =
convert10s(simout,simtimes,grafo_ALL,interval,mysimoutinterval,nodes,optout,Fals
e)

    mysimout.to_csv(Path(saveindir / 'Detailsim.csv'), index=False, sep=',')

    # Build reality
    realstatus = pd.DataFrame(columns=mysimout.columns)
    # In order to build reality we start from the simulated path and every 10
seconds a random process is
    # responsible to introduce a delay. In that case 10 seconds are lost and the
remaining steps are all delayed.
    # Number of vehicles:
    nvehicles = mysimout['Vehicle'].unique().tolist()
    ##file for web filtered on vehicles active
    #### RICCARDO
    fleet_web=Fleet.loc[Fleet['VehicleID'].isin(nvehicles)]
    #### RICCARDO FINE
    for i in nvehicles:

        leggiout = True
        timerecovery = mysimout['SimTime'][mysimout['Vehicle'] == i].values[0]
        first = True

        while leggiout:
            if first:
                realtemp = mysimout[mysimout['Vehicle'] == i]
                opt = optout[optout['VehicleID'] == i]
            else:
                realtemp = mysimoutrec
                opt = optoutrec[optoutrec['VehicleID'] == i]
            delay = 0; SOCw = 0
            mytimelist = realtemp['SimTime'].tolist()

            for ii in mytimelist[mytimelist.index(timerecovery):-1]:
                leggiout = False
                line = realtemp[realtemp['SimTime'] == ii].copy(deep=True)
                line['SimTime'] = line['SimTime'] + delay
                line['SOC'] = line['SOC'] - SOCw
                realstatus = realstatus.append(line)
                if random.random() > randthres:
                    line['SimTime'] = line['SimTime'] + mysimoutinterval
                    delay = delay + mysimoutinterval
                    line['SOC'] = line['SOC'] - SOC10s
                    SOCw = SOCw + SOC10s
                    realstatus = realstatus.append(line)
                if realtemp['SimTime'][realtemp['SimTime'] == ii].values[0] /
```

```
300. == int(
                                realtemp['SimTime'][realtemp['SimTime'] ==
ii].values[0] / 300.):
                        # check if I am in a recharge
                        riga0 = line['Delivered'].values[0] + 1
                        #print(riga0)
                        opttemp = opt.head(riga0)
                        nzero = (opttemp['Deliveries'] == 0).sum()
                        #print('nzero: ' +str(nzero))
                        line1 = riga0 + nzero - 2 # Non chiaro perche qui ci vuole
un -2, uno è per inizio, altro per riga prec?
                        if len(opt.index) > line1:
                            mytemp1 = opt.iloc[line1,5]
                            if len(opt.index) > line1 + 2:
                                mytemp2 = opt.iloc[line1 + 1, 5]
                                if len(opt.index) > line1 + 2:
                                    mytemp3 = opt.iloc[line1 + 2, 5]
                                else:
                                    mytemp3 = mytemp2
                            else:
                                mytemp2 = mytemp1; mytemp3 = mytemp1
                        else:
                            mytemp1 = opt.iloc[riga0 + nzero,5]; mytemp2 = mytemp1;
mytemp3 = mytemp1
                        #print('Mytemp 1,2,3: ' + str(mytemp1) + ' ' + str(mytemp2)
+ ' ' + str(mytemp3))
                        if mytemp1 == 0 or mytemp2 == 0 or mytemp3 == 0:
                            allow = False
                        else:
                            allow = True
                        if line['Node'].values[0] in opt['DeliveryPointID'].values:
                            allow = False
                            #print('Attenzione sono un delivery point: ' +
str(line['Node'].values[0]))
                        soct = socabs * Fleet['maxCharge'][Fleet['VehicleID'] ==
i].values[0]
                        if SOCw / line['SOC'].values[0] > socratio and
line['SOC'].values[0] < soct and allow:
                            # Wait line Simtime to get to a multiple of 300...
                            nministep = int((300 - line['SimTime'].values[0] % 300)
/ 10)
                            for j in range(nministep):
                                line['SimTime'] = line['SimTime'] + mysimoutinterval
                                realstatus = realstatus.append(line)
                            # launch recovery
                            # Create Update_Veh.csv
                            data = {'VehicleID': [i],'timeStamp':
[line['SimTime'].values[0] / 60],
                                    'Latitude':[line['Latitude'].values[0]],
'Longitude':[line['Longitude'].values[0]],

'Node':[line['Node'].values[0]],'SOC':[line['SOC'].values[0]],

'Dist':[line['Dist'].values[0]],'lastDel':[line['Delivered'].values[0]+nzero-1]}
                            vehiclerec = pd.DataFrame(data)
                            vehiclerec.to_csv(Path(saveindir / 'Update_Veh.csv'),
index=False, sep=',')
                            # Temporary remove any charging station already in opt:
this is useless since
                            # colonnine is not an input of recovery
                            #colonnine['Remove'] = colonnine.apply(lambda x:
```

```
nocsinopt(opt, x['Charging Station ID']),
                        #                                          axis=1)
                        #colonnine = colonnine.drop(colonnine[colonnine.Remove
== True].index)
                        #colonnine.drop(columns=['Remove'], inplace=True)
                        #colonnine.to_csv(Path(saveindir /
'ChargingStations.csv'),index=False,sep=';')

                        # Launch
                        origdir = os.getcwd()
                        os.chdir(matlabdir)
                        myproc = subprocess.Popen([Path(matlabdir /
'run_Recovery.sh'),Path(matlabruntime)])
                        #myproc = subprocess.Popen([Path(matlabdir /
'run_Update.sh'),Path(matlabruntime)])
                        myproc.wait()
                        print('Fatto Recovery')
                        os.chdir(origdir)
                        timerecovery = line['SimTime'].values[0]
                        print('Timerecovery: ' + str(timerecovery))
                        leggiout = True
                        first = False
                        # Read Recovery Output
                        simouttmprec = pd.read_csv(Path(saveindir /
'simOut_rec.csv'),delimiter=',')
                        simoutrec = simouttmprec.iloc[:,:7]
                        optoutrec = pd.read_csv(Path(saveindir / 'OptOut.csv'),
delimiter=',')
                        # Rename new optout and simout
                        optnewname = 'OptOut_rec_' + str(i) + '_' +
str(timerecovery) + '.csv'
                        os.rename(Path(saveindir / 'OptOut.csv'), Path(saveindir
/ optnewname))

timestep.append({'type':'rec','time':timerecovery,'vehicleid':i})

                        ##merge lat lon with deliverypoints for web
                        ## RICCARDO INIZIO
                        json_new_name = 'coord_'+str(optnewname)+'.json'
                        genera_file_opt_out_coord(Path(saveindir /
optnewname),Path(saveindir / 'Nodes.csv'),Path(saveindir / json_new_name ))
                        ## RICCARDO FINE

                        simnewname = 'simOut_rec_' + str(i) + '_' +
str(timerecovery) + '.csv'
                        os.rename(Path(saveindir / 'simOut_rec.csv'),
Path(saveindir / simnewname))
                        plannewname = 'planOut_rec_' + str(i) + '_' +
str(timerecovery) + '.mat'
                        shutil.copy2(Path(saveindir / 'planOut.mat'),
Path(saveindir / plannewname))

                        simtimes = simoutrec['SimTime'][simoutrec['Vehicle'] ==
simoutrec['Vehicle'].values[0]].tolist()
                        mysimoutrec =
convert10s(simoutrec,simtimes,grafo_ALL,interval,mysimoutinterval,nodes,optoutre
c,False)
                        #realstatus.to_csv(Path(matlabdir / 'Realstatus.csv'),
index=False, sep=',')
                        detsimnewname = 'Detailsim_rec_' + str(i) + '_' +
```

```
str(timerecovery) + '.csv'
                        mysimoutrec.to_csv(Path(saveindir / detsimnewname),
index=False, sep=',')
                        updatevehname = 'Update_Veh_' + str(i) + '_' +
str(timerecovery) + '.csv'
                        shutil.copy2(Path(saveindir / 'Update_Veh.csv'),
Path(saveindir / updatevehname))
                        # Cut the last line of realstatus in order to avoid
duplicates
                        realstatus.drop(index=realstatus.index[-
1],axis=0,inplace=True)
                        break

                    timeref = ii + delay
                    if timeref < realtemp['SimTime'].values[-1]:
                        timeref = ii+delay
                    else:
                        timeref = realtemp['SimTime'].values[-1]
                    #print('timeref: ' + str(timeref) + ', ii: ' + str(ii) + ',
delay: ' + str(delay))
                    if line['Delivered'].values[0] -
realtemp['Delivered'][realtemp['SimTime'] == timeref].values[0] < deldelay and
allow:
                        # Wait line Simtime to get to a multiple of 300...
                        nministep = int((300 - line['SimTime'].values[0] % 300)
/ 10)
                        for j in range(nministep):
                            line['SimTime'] = line['SimTime'] + mysimoutinterval
                            realstatus = realstatus.append(line)
                        # launch update
                        # Create Update_Veh.csv
                        data = {'VehicleID': [i], 'timeStamp':
[line['SimTime'].values[0] / 60],
                                'Latitude': [line['Latitude'].values[0]],
'Longitude': [line['Longitude'].values[0]],
                                'Node': [line['Node'].values[0]], 'SOC':
[line['SOC'].values[0]],
                                'Dist': [line['Dist'].values[0]], 'lastDel':
[line['Delivered'].values[0] + nzero - 1]}
                        vehiclerec = pd.DataFrame(data)
                        vehiclerec.to_csv(Path(saveindir / 'Update_Veh.csv'),
index=False, sep=',')
                        # Create Update Links.csv, only used in Update
                        newlinks = links[['ID', 'Speed (00:00-00-
59)']].copy(deep=True)
                        newlinks['Speed'] = np.random.randint(-20, 20,
newlinks.shape[0])
                        newlinks['Speed'] = newlinks['Speed'] + newlinks['Speed
(00:00-00-59)']
                        newlinks['Speed'].where(newlinks['Speed'] >= 10, 10,
inplace=True)
                        newlinks.drop('Speed (00:00-00-59)', axis=1,
inplace=True)
                        newlinks.to_csv(Path(saveindir / 'Update_Links.csv'),
index=False, sep=',')

                        # Launch
                        origdir = os.getcwd()
                        os.chdir(matlabdir)
                        myproc = subprocess.Popen([Path(matlabdir /
'run_Update.sh'),Path(matlabruntime)])
```

```
                        myproc.wait()
                        print('Fatto Update')
                        os.chdir(origdir)
                        timerecovery = line['SimTime'].values[0]
                        print('Timeupdate: ' + str(timerecovery))
                        leggiout = True
                        first = False
                        # Read Update Output
                        simouttmprec = pd.read_csv(Path(saveindir /
'simOut_upd.csv'), delimiter=',')
                        simoutrec = simouttmprec.iloc[:, :7]
                        optoutrec = pd.read_csv(Path(saveindir / 'OptOut.csv'),
delimiter=',')
                        # Rename new optout and simout
                        optnewname = 'OptOut_upd_' + str(i) + '_' +
str(timerecovery) + '.csv'
                        os.rename(Path(saveindir / 'OptOut.csv'), Path(saveindir
/ optnewname))

                        ##RICCARDO INIZIO

timestep.append({'type':'upd','time':timerecovery,'vehicleid':i})
                        ##merge lat lon with deliverypoints for web
                        json_new_name = 'coord_'+str(optnewname)+'.json'
                        genera_file_opt_out_coord(Path(saveindir /
optnewname),Path(saveindir / 'Nodes.csv'),Path(saveindir / json_new_name ))
                        ##RICCARDO FINE

                        simnewname = 'simOut_upd_' + str(i) + '_' +
str(timerecovery) + '.csv'
                        os.rename(Path(saveindir / 'simOut_upd.csv'),
Path(saveindir / simnewname))
                        plannewname = 'planOut_upd_' + str(i) + '_' +
str(timerecovery) + '.mat'
                        shutil.copy2(Path(saveindir / 'planOut.mat'),
Path(saveindir / plannewname))

                        simtimes = simoutrec['SimTime'][simoutrec['Vehicle'] ==
simoutrec['Vehicle'].values[0]].tolist()
                        mysimoutrec = convert10s(simoutrec, simtimes, grafo_ALL,
interval, mysimoutinterval, nodes,
                                                 optoutrec, False)
                        detsimnewname = 'Detailsim_upd_' + str(i) + '_' +
str(timerecovery) + '.csv'
                        mysimoutrec.to_csv(Path(saveindir / detsimnewname),
index=False, sep=',')
                        updatevehname = 'Update_Veh_' + str(i) + '_' +
str(timerecovery) + '.csv'
                        shutil.copy2(Path(saveindir / 'Update_Veh.csv'),
Path(saveindir / updatevehname))
                        updatelinksname = 'Update_Links_' + str(i) + '_' +
str(timerecovery) + '.csv'
                        shutil.copy2(Path(saveindir / 'Update Links.csv'),
Path(saveindir / updatelinksname))
                        # Cut the last line of realstatus in order to avoid
duplicates
                        realstatus.drop(index=realstatus.index[-1], axis=0,
inplace=True)
                        break
```

```
    ##RICCARDO INIZIO
    ## generate file for web Fleet and STEP
    DF_fleet_web = pd.DataFrame(fleet_web)
    DF_fleet_web.to_csv(Path(saveindir / 'Fleet_web.csv'), index=False, sep=',')
    DF_timestep = pd.DataFrame(timestep)
    DF_timestep.to_csv(Path(saveindir / 'Step.csv'), index=False, sep=',')
    ###RICCARDO FINE

    realstatus.to_csv(Path(saveindir / 'Realstatus.csv'), index=False, sep=',')

###prendo file da matllab opt e unisco lat lon da delivery points

def genera_file_opt_out_coord(fileopt,nodes,output_file):

    result=[]

    with open(fileopt,'r')as fcd:
        fcd_reader= csv.DictReader(fcd)
        fcd = list( fcd_reader)
        for fcdrow in fcd:
            with open(nodes,'r')as fcd1:
                fcd_reader1= csv.DictReader(fcd1,delimiter=';')
                fcd1 = list( fcd_reader1)
                #print(fcd1)
                filtered = list(filter(lambda p: (p['ID_Node'] ==
str(fcdrow["DeliveryPointID"])), fcd1))
                #print(filtered)
                #print(fcdrow["DeliveryPointID"])
                if (int(fcdrow['Deliveries'])==0 and
int(fcdrow['VehicleLoad'])>0):
                    type_stop="recharge"
                else:
                    type_stop="delivery"

                for ff in filtered:
                    result.append({'VehicleID':fcdrow['VehicleID'],
'DepartureTime':fcdrow['DepartureTime'],
                    'TravelledDistance':fcdrow['TravelledDistance'] ,
                    'Deliveries': fcdrow['Deliveries'],
                    'ArrivalTime': fcdrow['ArrivalTime'],
                    'DeliveryPointID': fcdrow['DeliveryPointID'],
                    'VehicleLoad':fcdrow['VehicleLoad'],
                    'TypeStop':type_stop,
                    'SOC':fcdrow['SOC'],
                    'lat':ff['ycoord'],
                    'lon':ff['xcoord']
                    })
        with open(output_file, 'w') as outfile:
            json.dump(result, outfile)


def
convert10s(simoutl,simtimes,grafo ALL,interval,mysimoutinterval,nodes,optoutl,pe
rccons):
    # Ciclo su simtimes
    # Per quanto riguarda le distanze percorse utilizzo la libreria osmnx con
individuazione dei nodi
    # intermedi e calcolo delle distanze fra i nodi.
    # Invece per il calcolo del SOC la scelta e' di non legarsi al calcolo dei
consumi matlab
    # discutibile e comunque soggetto a cambiamenti, ma di fare una
```

```
interpolazione lineare pesata
    # dalle distanze percorse.
    mysimout = pd.DataFrame(columns=simoutl.columns)
    for veh in simoutl['Vehicle'].unique():
        for step in simtimes:
            presentnode = simoutl['Node'][(simoutl['Vehicle'] == veh) &
(simoutl['SimTime'] == step)].values[0]
            if step == simtimes[0]:
                previousnode = presentnode
                previousstep = step
                socold = simoutl['SOC'][(simoutl['Vehicle'] == veh) &
(simoutl['SimTime'] == step)].values[0]
            if step != simtimes[0] and presentnode != previousnode:
                route = nx.shortest_path(G=grafo_ALL, source=previousnode,
                        target=presentnode, weight='travel_time')
                trtimes = ox.utils_graph.get_route_edge_attributes(grafo_ALL,
route, attribute='travel_time',

minimize_key='travel_time')
                trtimes[:] = [x * interval / sum(trtimes) for x in trtimes]
                trdist = ox.utils_graph.get_route_edge_attributes(grafo_ALL,
route, attribute='length',

minimize_key='travel_time')
                #print(trtimes); print(trdist);print(route)

                tempsimout = simoutl[(simoutl['Vehicle'] == veh) &
(simoutl['SimTime'] == previousstep)].copy(deep=True)
                tempsimout['SimTime'] = tempsimout['SimTime'] * 60
                # Handle SOC in case of recharge and negative difference
                mysimout = mysimout.append(tempsimout)
                realtime = previousstep * 60
                nodetime = previousstep * 60
                disttmp = simoutl['Dist'][(simoutl['Vehicle'] == veh) &
(simoutl['SimTime'] == previousstep)].values[0]
                ii = 0
                prevnode = route[ii]
                while realtime < step * 60 - mysimoutinterval:
                    realtime = realtime + mysimoutinterval
                    '''if realtime >= nodetime + trtimes[-1]:
                        print('Attenzione: ho usato tutti i trtimes')
                        tempsimout['SimTime'] = realtime
                        mysimout = mysimout.append(tempsimout)
                        print(tempsimout)
                    elif realtime < nodetime + trtimes[ii]:'''
                    if realtime < nodetime + trtimes[ii]:
                        prevnode = route[ii]
                        tempsimout['SimTime'] = realtime
                        soctmp2 = simoutl['SOC'][(simoutl['Vehicle'] == veh) &
(simoutl['SimTime'] == step)].values[0]
                        if socold < soctmp2: # caso ricarica. Così non funziona
perchè sta ricaricando mentre è in marcia
                            timeend = step * 60
                            socold = socold + (soctmp2 - socold) / (timeend -
realtime)
                            tempsimout['SOC'] = socold
                        mysimout = mysimout.append(tempsimout)
                        #print(tempsimout)
                    else:
                        while nodetime + trtimes[ii] < realtime:
```

```
                            nodetime = nodetime + trtimes[ii]
                            ii=ii+1
                        nodetmp = route[ii-1]
                        lattemp = nodes['ycoord'][nodes['ID_Node'] ==
nodetmp].values[0]
                        lontemp = nodes['xcoord'][nodes['ID_Node'] ==
nodetmp].values[0]
                        soctmp1 = simoutl['SOC'][(simoutl['Vehicle'] == veh) &
(simoutl['SimTime'] == previousstep)].values[0]
                        soctmp2 = simoutl['SOC'][(simoutl['Vehicle'] == veh) &
(simoutl['SimTime'] == step)].values[0]
                        peso1 = sum(trdist[:ii-1]); peso2 = sum(trdist[ii:])
                        #print(str(peso1) + ' ' + str(peso2))
                        # Handle battery recharge in case of moving vehicle
                        if soctmp1 < soctmp2: soctmp1 = soctmp2
                        soctmp = soctmp1 - (soctmp1 - soctmp2) * peso1 / (peso1
+ peso2)
                        socold = soctmp
                        disttmp = disttmp +
(nx.shortest_path_length(G=grafo_ALL, source=prevnode,

target=nodetmp, weight='length')) / 1000.
                        tempsimout =
pd.DataFrame([[veh,realtime,lattemp,lontemp,nodetmp,soctmp,disttmp]],
                                                    columns=simoutl.columns)
                        #print(tempsimout)
                        mysimout = mysimout.append(tempsimout)
                        prevnode = nodetmp

            elif step != simtimes[0] and presentnode == previousnode:
                tempsimout = simoutl[(simoutl['Vehicle'] == veh) &
(simoutl['SimTime'] == previousstep)].copy(deep=True)
                tempsimout['SimTime'] = tempsimout['SimTime'] * 60
                #print(tempsimout)
                mysimout = mysimout.append(tempsimout)
                realtime = previousstep * 60
                while realtime < step * 60 - mysimoutinterval:
                    realtime = realtime + mysimoutinterval
                    tempsimout['SimTime'] = realtime
                    soctmp1 = simoutl['SOC'][(simoutl['Vehicle'] == veh) &
(simoutl['SimTime'] == step)].values[0]
                    soctmp2 = simoutl['SOC'][(simoutl['Vehicle'] == veh) &
(simoutl['SimTime'] == step)].values[0]
                    socold = soctmp1
                    if soctmp1 < soctmp2:
                        peso1 = realtime - previousstep * 60
                        peso2 = step * 60 - realtime
                        socold = soctmp1 + (soctmp2 - soctmp1) * peso1 / (peso1
+ peso2)
                        tempsimout['SOC'] = socold
                    mysimout = mysimout.append(tempsimout)
                    #print(tempsimout)

            previousnode = presentnode
            previousstep = step

    # Add a column with information about number of deliveries performed
    optoutl = optoutl[optoutl['Deliveries'] != 0]
    ndelivery = optoutl.groupby(['VehicleID'],as_index=False).agg({'Deliveries':
'count'})
    opttest = optoutl[optoutl['VehicleID'] == mysimout['Vehicle'].values[0]]
```

```
    opttest = opttest[opttest['ArrivalTime'].multiply(60) <
mysimout['SimTime'].values[50]]
    mysimout['Delivered'] = mysimout.apply(lambda x: deladvance(optoutl,
ndelivery, x['Vehicle'], x['SimTime'],perccons), axis=1)

    return mysimout

def deladvance(opto,ndel,vid,stime,perc):
    totdel = float(ndel['Deliveries'][ndel['VehicleID'] == vid].values[0])
    #print(totdel)
    optot = opto[opto['VehicleID'] == vid]
    optot = optot[optot['ArrivalTime'].multiply(60) < stime]
    if perc:
        delivered = float(len(optot.index) / totdel)
    else:
        delivered = int(len(optot.index))
    #print(delivered)
    return delivered

def calcnewid(grafo, lon, lat):
    # Get nearest node to the given point. Deprecated: to be changed with
distance.nearest.node
    #newid = ox.get_nearest_node(grafo, (lat, lon), method='euclidean')
    newid = ox.distance.nearest_nodes(grafo, lon, lat)
    return newid

def selectvel(ms,rt,fact):
    # Compute the velocity to be assigned to the considered link. If a list of
velocities is provided
    # compute the average, otherwise take the given value. If NAN is provided
check rt and choose the
    # best velocity.
    # if more velocities are present in line
    if isinstance(ms, list):
        #print(ms)
        myl = list(float(i) for i in ms)
        #vel = sum(myl) / len(myl) * fact
        vel = sum(myl) / len(myl)
    elif is_number(ms):
        #vel = float(ms) * fact
        vel = float(ms)
        if math.isnan(vel):
            if rt == 'residential':
                vel = 30
            else:
                vel = 50
    else:
        print('Error, case not allowed' + str(ms))
        exit()

    vel = vel * fact

    return int(vel)

def is_number(s):
    try:
        float(s)
        return True
    except ValueError:
        return False
```

```
def nocsindp(delivery,node):
    if node in delivery['Delivery Point ID'].values:
        remove = True
        print('Warning, removing charging station n: ' + str(node))
    else:
        remove = False
    return remove

def nocsinopt(opt,node):
    if node in opt['DeliveryPointID'].values:
        remove = True
        print('Warning, removing charging station n: ' + str(node))
    else:
        remove = False
    return remove

def nopeso(w):
    if w == 0:
        remove = True
        print('Warning, removing delivery point')
    else:
        remove = False
    return remove


if __name__ == '__main__':
    optim_in()
```