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

Ministero della Transizione Ecologica

Ricerca di Sistema elettrico

Protocolli di comunicazione IOT per la gestione di flotte elettriche

V. Rosato, M. Pollino, G. Tomasino, F. Ortenzi (ENEA)

Report RdS/PTR(2020)/058

PROTOCOLLI DI COMUNICAZIONE IOT PER LA GESTIONE DI FLOTTE ELETTRICHE

V. Rosato, M. Pollino, G. Tomasino, F. Ortenzi (ENEA)

Aprile 2021

Report Ricerca di Sistema Elettrico

Accordo di Programma Ministero dello Sviluppo Economico - ENEA
Piano Triennale di Realizzazione 2019-2021 - II annualità
Obiettivo: Tecnologie
Progetto: Tecnologie per la penetrazione efficiente del vettore elettrico negli usi finali
Work package: Mobilità
Linea di attività: 2.20 - Sviluppo di moduli di calcolo e simulazione della mobilità urbana e di protocolli di comunicazione IoT per la gestione real-time di flotte elettriche
Responsabile del Progetto: Claudia Meloni, ENEA
Responsabile del Work package: Maria Pia Valentini, ENEA

Ringraziamenti: Fabio Carapellucci (ENEA – Sistemi e Tecnologie per la Mobilità Sostenibile - TERIN PSU STMS)

Indice

1. Sommario

Le attività descritte nel presente Report riguardano la seconda annualità del Piano Triennale di Realizzazione 2019-2021, Progetto "Tecnologie per la penetrazione efficiente del vettore elettrico negli usi finali".

In particolare, esse sono relative alla Linea di Attività LA2.20 "*Sviluppo di moduli di calcolo e simulazione della mobilità urbana e di protocolli di comunicazione IoT per la gestione real time di flotte elettriche*" del Work package WP2.

Come da programma, sono stati messi a punto i protocolli di comunicazione fra i diversi componenti della distribuzione urbana delle merci, in grado di fornire all'operatore logistico un monitoraggio in tempo reale dei parametri essenziali per la gestione delle consegne e di eventuali criticità nella disponibilità di energia a bordo dei veicoli (monitoraggio dello stato di carica della batteria attraverso acquisizione dati dal Can bus) e a terra (monitoraggio della disponibilità dei punti di ricarica attraverso l'acquisizione dei dati remoti di occupazione del parcheggio resi disponibili tramite l'utilizzo di apposita sensoristica)

Tutti i dati raccolti vengono inviati in tempo reale alla piattaforma software "eMu" [1], sviluppata e sperimentata nell'area metropolitana di Roma nel corso del periodo di ricerca relativo al PAR 2015-18, come strumento di ausilio allo studio di scenari di diffusione di flotte di veicoli elettrici nelle aree urbane.

Le stesse informazioni vengono anche inviate alla piattaforma di supporto alle decisioni "CIPCast-ER" [2], lo strumento per l'analisi e la protezione delle infrastrutture critiche, sviluppato nel periodo di ricerca relativo al PAR 2013-2015.

2. Introduzione

Per far fronte all'aggravarsi dei fenomeni urbani di congestione, inquinamento e consumo energetico, la soluzione che sembra prospettarsi da qualche anno, messa in atto all'interno delle realtà cittadine e metropolitane, prevede la sostituzione dell'attuale flotta di veicoli a combustione interna, con veicoli elettrici di pari dimensioni e caratteristiche.

Al fine però di ottimizzare l'intero processo e i risultati ottenuti, occorre effettuare una perfetta pianificazione integrata trasporti-territorio, per gestire al meglio le flotte elettriche, i tragitti e le ricariche.

Negli ultimi anni, gli studi di settore e le attività di ricerca correlate, si sono quindi concentrati sul progetto di moderne piattaforme di supporto alle decisioni [3] per amministratori locali e operatori del settore, volte ad individuare, mediante processi analitici, politiche e interventi atti a favorire la transizione verso un modello di Smart Mobility energeticamente efficiente e a bassa intensità di carbonio.

In questo scenario, rivestono importanza fondamentale le informazioni che è possibile acquisire in tempo reale, sia dal veicolo in movimento, che dal territorio che esso è chiamato a percorrere.

Per quanto riguarda il veicolo, le informazioni di interesse sono tutte quelle inerenti la posizione istantanea e lo stato di carica della batteria.

Ciò al fine di allertare il *sistema di controllo delle flotte*, riguardo alla eventualità *di ritardi rispetto alla programmazione ex-ante del viaggio (con particolare riferimento a missioni di consegna merci ad orario prestabilito)* o ad insufficiente autonomia del veicolo, che potrebbe comportare il fallimento della missione, non potendo in questo caso il veicolo completare il giro assegnato all'inizio del percorso.

In questi casi la piattaforma intelligente dovrà rielaborare il percorso in modo da limitare i ritardi nelle consegne (eventualmente tenendo conto delle impreviste condizioni di traffico sulla rete, disponendo di un sistema di acquisizione di tale informazione) e/o inviare il veicolo alla più vicina stazione di ricarica presente sul territorio, avendo però cura di verificare che essa abbia postazioni libere e possa quindi ospitare il veicolo, e che non sia invece occupata in tutte le sue postazioni.

Questa è l'altra informazione che la piattaforma di controllo deve acquisire sul territorio, ovvero la disponibilità o meno delle postazioni di ricarica.

Tale informazione deve essere inoltre aggiornata ad intervalli regolari, affinché si abbia sempre la completa conoscenza della mappa delle postazioni di ricarica disponibili, per poterne usufruire a seconda delle necessità.

3. Acquisizione dati dal veicolo: stato dell'arte

3.1 Lo Standard OBDII

Il protocollo di riferimento universalmente utilizzato per la diagnostica di un veicolo, ovvero per acquisire tutti i dati di interesse dal veicolo, è **OBD** (***On Board Diagnostic system***), sviluppato inizialmente in California dalla CARB (California Air Resources Board[1]) intorno agli anni 70 per monitorare e verificare che le emissioni del motore dei veicoli in circolazione fossero confacenti allo standard definito dall'Agenzia Nazionale Americana per la protezione dell'Ambiente (EPA - US Environmental Protection Agency[2]). Il protocollo OBD si è poi evoluto nella sua versione II ed è diventato obbligatorio negli Stati Uniti d'America per gran parte dei veicoli di nuova costruzione, a partire dal 1996. Successivamente, per passi successivi sulle varie categorie di veicoli, in un processo che si è concluso nel 2010, l'utilizzo del protocollo OBDII è stato esteso a tutti i veicoli di nuova costruzione. Contemporaneamente, il protocollo OBDII è stato formalizzato dal SAE (Society of Automated Engineers[3]) che ha prodotto diversi standard in cui vengono definite le caratteristiche dei protocolli di comunicazione, la struttura minima dei frame, i codici di errore, la struttura del connettore fisico, gli standard operativi minimi per gli strumenti di scansione e molto altro.

Il protocollo OBDII [4][5] nella sua versione definitiva, è ormai in grado di fornire un controllo quasi completo di tutte le parti del motore, e degli altri dispositivi elettronici presenti a bordo del veicolo.

Esso opera grazie allo scambio di codici univoci, denominati PIDs (Parameter IDs), che sono usati per riconoscere ed interpretare i comandi e le informazioni richieste. Di solito un PID è un codice formato da 1 o 2 byte, mentre invece le informazioni associate a ciascun PID, possono avere lunghezza variabile. Il caso più comune è la lunghezza di 4 byte, ma a seconda del PID specificato, le informazioni possono occupare anche 8, 16 o più bytes. Naturalmente ogni bit esprime un significato diverso a seconda del PID e della posizione che occupa, può trattarsi ad esempio di una velocità, di una pressione, di una temperatura, ecc..

In genere l'informazione risultante è costituita da un numero intero o reale, con o senza segno.

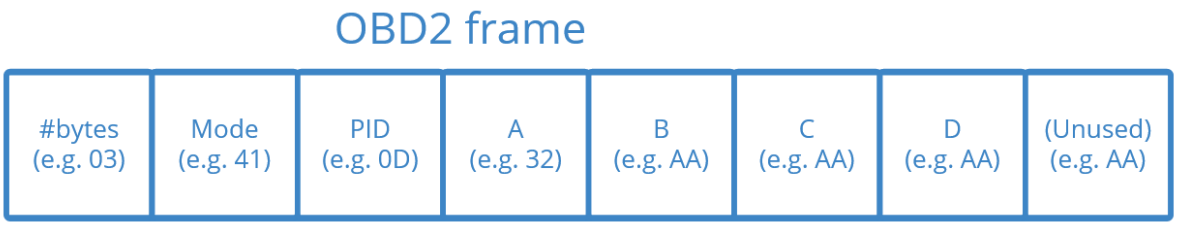


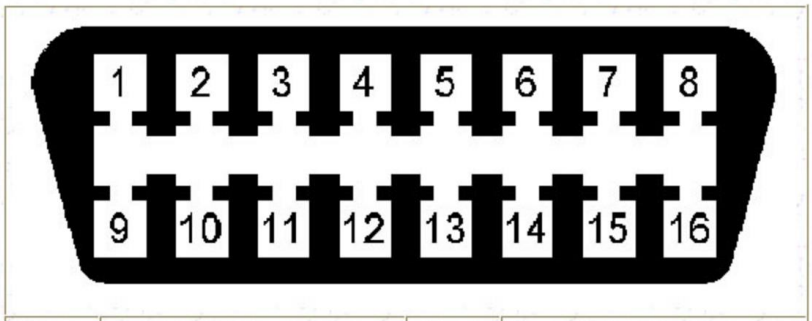
Figura 1 - Struttura del frame OBDII

[1] https://ww2.arb.ca.gov/about
[2] https://www.epa.gov/state-and-local-transportation/vehicle-emissions-board-diagnostics-obd
[3] https://www.sae.org

I principali costruttori di veicoli comunque, hanno ampliato l'insieme dei PID standard, introducendo un gran numero di ulteriori PIDs proprietari, utilizzati (e riconosciuti) solo da quelle particolari case costruttrici.

Anche la diagnostica dei malfunzionamenti del veicolo viene resa facile e immediata, grazie alla standardizzazione dei codici di errore, i cosiddetti DTCs (Diagnostic Trouble Codes) che il sistema OBD rende disponibili all'esterno del veicolo in caso di guasto.

La figura seguente mostra il pinout della presa OBD presente sul veicolo. Da notare che i Pin 6 e 14 garantiscono il collegamento con un eventuale dispositivo CAN BUS all'interno del veicolo.



PIN	DESCRIPTION	PIN	DESCRIPTION
1	Vendor Option	9	Vendor Option
2	J1850 Bus +	10	j1850 BUS
3	Vendor Option	11	Vendor Option
4	Chassis Ground	12	Vendor Option
5	Signal Ground	13	Vendor Option
6	CAN (J-2234) High	14	CAN (J-2234) Low
7	ISO 9141-2 K-Line	15	ISO 9141-2 Low
8	Vendor Option	16	Battery Power

Figura 2 - Pinout standard della presa OBDII

3.2 Il protocollo CAN BUS

Affinchè poi tutti i dispositivi a bordo del veicolo possano comunicare tra loro, scambiarsi informazioni e quant'altro, attraverso l'utilizzo di un Bus condiviso, è stato messo a punto (dalla Bosch intorno alla metà degli anni 80) un ulteriore protocollo, denominato **CAN** (***Controller Area Network***) [6][7], che definisce le modalità di comunicazione tra le centraline e gli altri dispositivi elettronici all'interno del veicolo.

Tale sistema indicato con la sigla CAN Bus, dal 1993 diventa standard internazionale (ISO 11898) nella sua versione estesa CAN Bus 2.0. Inoltre dal 2008 è lo standard di riferimento (ISO 15765) per la trasmissione dei frame del protocollo OBDII all'interno dei veicoli.

Il protocollo implementa la comunicazione attraverso i primi due livelli ISO/OSI, Physical Layer e Data Link Layer. I livelli superiori sono resi disponibili agli sviluppatori per ulteriori implementazioni.

Quindi, mentre il protocollo OBDII decodifica il significato delle informazioni all'interno del veicolo, CAN Bus è invece un protocollo di comunicazione, che definisce le modalità di trasporto delle informazioni all'interno del veicolo. I due protocolli quindi si completano perfettamente, ed insieme costituiscono la perfetta tecnologia di comunicazione uomo-veicolo.

Più in dettaglio il protocollo CAN Bus [8] opera realizzando una rete di comunicazione tra tutte le ECU's (Electronic Control Units) del veicolo. In questo modo, in fase di progettazione del veicolo, è possibile ottimizzare la presenza delle ECU, sia in quantità che per posizionarle nei punti critici del veicolo. Ciascuna di essa rileverà soltanto le informazioni di propria competenza, e le trasmetterà in rete alle altre ECU, utilizzando proprio il protocollo CAN Bus. Esempi di ECU sono l'unità di controllo del motore, degli airbags, del sistema audio, ecc..

Nelle auto moderne il numero di unità di controllo può essere anche dell'ordine del centinaio.

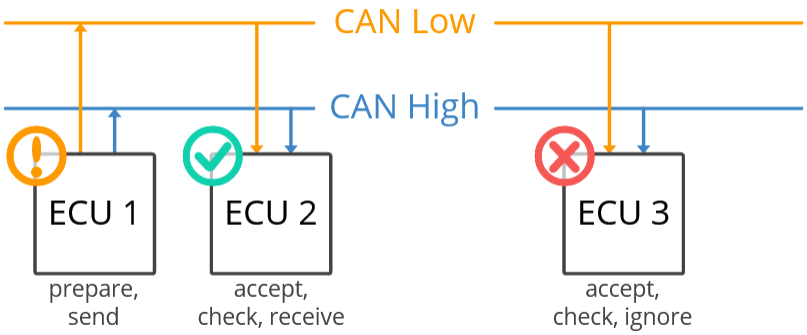


Figura 3-Funzionamento del CAN Bus

I vantaggi fondamentali della tecnologia CAN BUS sono i seguenti:

1. Economicità: la soluzione impiegata prevede un unico canale di comunicazione per tutte le ECU a bordo del veicolo, e ciò consente di ridurre notevolmente il costo di realizzazione, il peso e lo spazio ad esso destinato (l'alternativa molto più dispendiosa, sarebbe quella di prevedere un canale di comunicazione per ciascuna coppia di ECU)
2. Centralità: il sistema permette una gestione centralizzata di tutte le ECU, ed una veloce individuazione e correzione dei malfunzionamenti e dei guasti
3. Robustezza: il sistema è robusto ai disturbi elettrici e alle interferenze elettromagnetiche
4. Efficienza: i messaggi del CAN Bus possono essere classificati per tipologia e inoltrati secondo priorità
5. Flessibilità: ogni ECU può ricevere i messaggi di tutte le altre ECU e decidere cosa tenere, cosa scartare e comportarsi di conseguenza
6. Scalabilità: Il sistema è facilmente scalabile, perché è possibile facilmente aggiungere ECU al canale di comunicazione.

Il frame di comunicazione del protocollo CAN Bus 2.0A (quello adottato dalla maggior parte delle automobili) prevede l'utilizzo di 11 bit per il campo identificatore del frame (ID), e 64 bit per i dati di informazione.
Inoltre 16 bit sono utilizzati per effettuare un controllo di ridondanza.

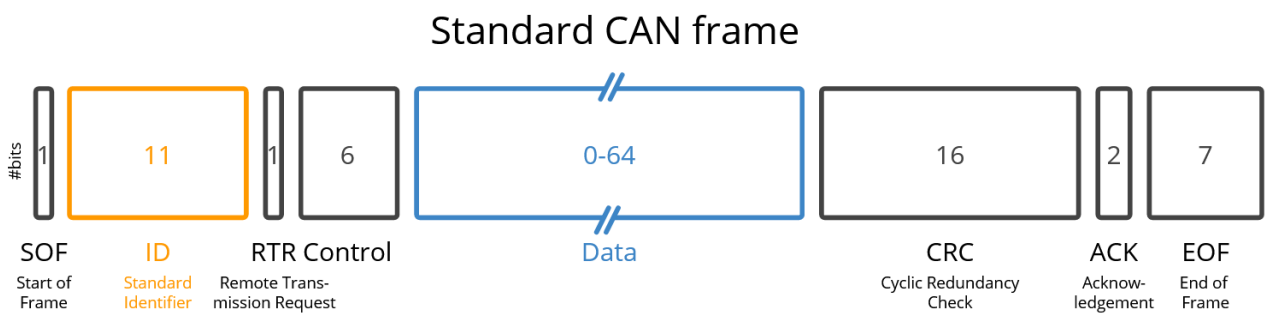


Figura 4- Struttura del frame CAN Bus

3.3 Il sistema di acquisizione dati

Come accennato in precedenza, la necessità di acquisire in tempo reale dei dati da un veicolo in movimento, non è una novità ed è stato già affrontato in diversi ambiti e per perseguire svariati obiettivi. Spesso ad esempio, l'obiettivo primario è stato quello di indagare sulle emissioni [9][10], ma nel tempo è stata messa a punto un'apposita tecnologia finalizzata ad acquisire in tempo reale molti dati dal veicolo.

Tale tecnologia comprende necessariamente la presenza a bordo di una cosiddetta "Onboard Unit", ovvero di uno strumento finalizzato all'acquisizione dei dati del veicolo. Tale strumento deve essere in grado di interfacciarsi con il CAN Bus, per scambiare messaggi al fine di acquisire le informazioni necessarie.

Tutto ciò viene egregiamente realizzato da un dispositivo denominato CAN-USB, che possiede una porta di collegamento al veicolo, e un'altra di tipo seriale USB, che permette di trasferire i dati ad un sistema di controllo, tipicamente un PC o un Notebook, collegato al dispositivo CAN-USB proprio attraverso la porta USB.

Sul sistema di controllo, al fine di gestire la comunicazione tra i diversi sistemi, è in esecuzione un software apposito, che in genere è ampiamente personalizzabile attraverso l'utilizzo di apposite API messe a disposizione dalla casa produttrice del dispositivo CAN-USB. Ciò consente di scegliere ad esempio, quali dati acquisire, la frequenza di acquisizione, nonché la modalità di archiviazione di tali dati.

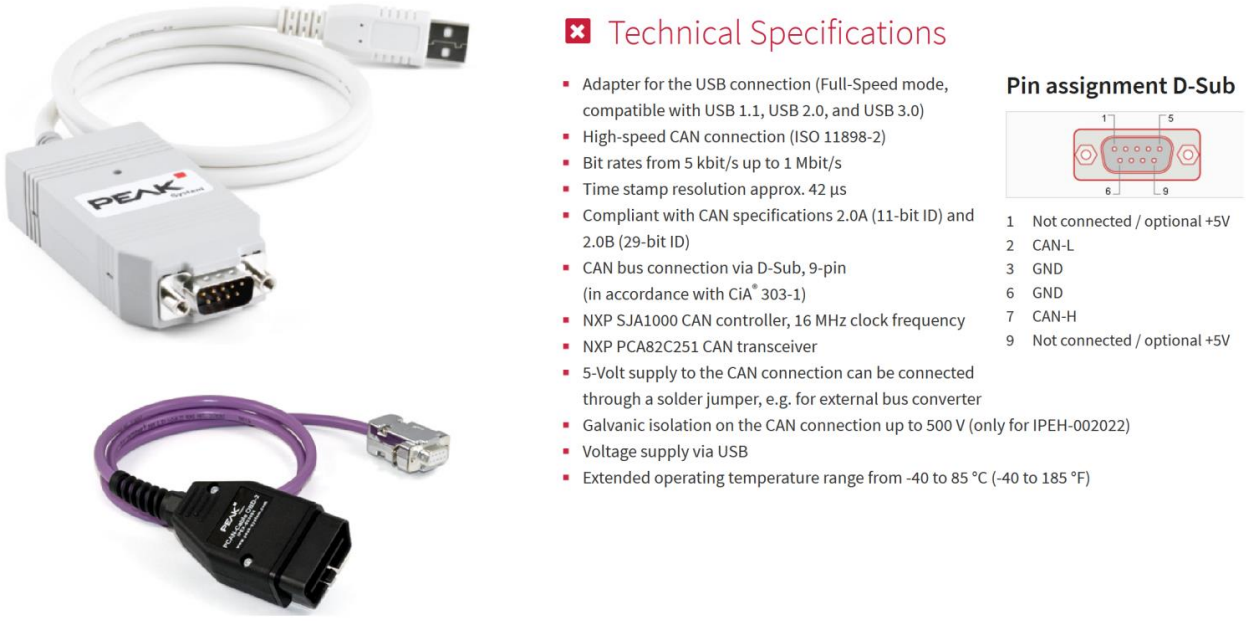


Figura 5- Il dispositivo CAN-USB

Come si vede in Fig.5, il dispositivo CAN-USB possiede una porta seriale DB-9, che può essere collegata tramite l'apposito adattatore (di colore viola in figura), alla presa OBD del veicolo. Dall'altro lato invece, la porta USB di uscita del dispositivo sarà collegata al PC o al Notebook su cui verrà eseguito il software per acquisire i dati di interesse. Il software in questione utilizzerà le API messe 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.

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.
E' 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, come spiegheremo meglio nel prossimo capitolo, la nostra scelta è ricaduta sui moderni dispositivi Raspberry PI[4], sistemi di piccole dimensioni, facilmente trasportabili, ma che dal punto di vista delle prestazioni sono paragonabili a PC (o Notebook) di ultima generazione.

4. La soluzione proposta

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.
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.

[4] https://www.raspberrypi.org/

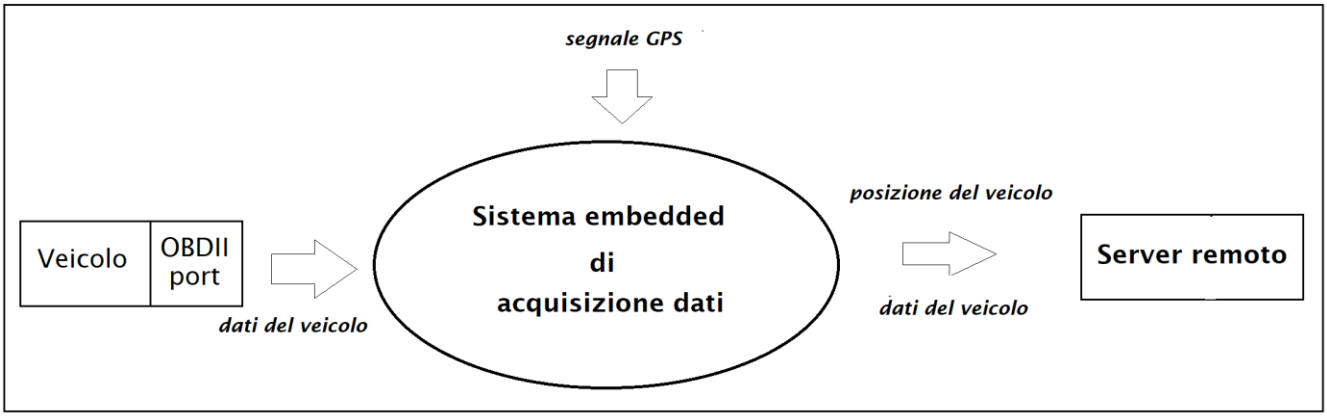


Figura 6 - Il sistema embedded di acquisizione dati

Per la realizzazione del *sistema embedded* si è utilizzato un dispositivo Raspberry PI 4B, al quale 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.

L'interfaccia prescelta è la PCAN-USB della Peak System[5], che possiede un ottimo rapporto tra qualità del prodotto e prezzo di acquisto.

Il sistema completo è illustrato nello schema seguente.

[5] Home Page Peak System

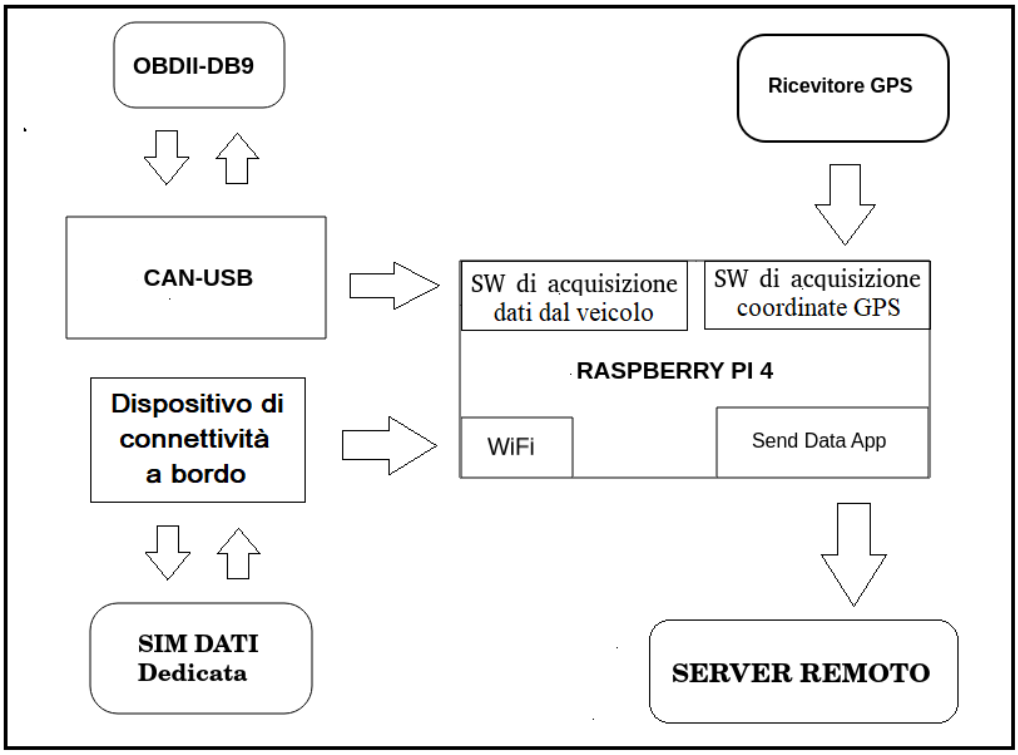


Figura 7- Gli elementi del sistema embedded

4.1 Il Raspberry PI

Il Raspberry PI [11] è un sistema calcolatore a scheda singola (SBC, Single Board Computer), progettato in UK dalla Raspberry Foundation[6] agli inizi degli anni 2000, anche se le prime schede furono diffuse solo a partire dal 2012. Ideato inizialmente per scopi didattici, si è molto diffuso negli ultimi anni, grazie alla capacità di offrire prestazioni elevate, ma anche allo stesso tempo, grande compattezza ed economicità.

Il Raspberry PI infatti, è poco più grande di un pacchetto di sigarette (8cm x 5cm circa), e nella sua versione più performante, possiede un processore ARM a 64 bit da 1.5GHz, RAM fino a 8GB, 4 porte USB (di cui 2 USB 3.0), una porta Gigabit Ethernet, due porte HDMI, connettività Bluetooth e Wireless, uno slot per la memoria SD Card in cui va installato il Sistema Operativo ed eventuali dati utente. Il Sistema Operativo utilizzato è chiamato Raspbian, ed è basato sulla distribuzione Debian del Sistema Operativo Linux.

[6] https://www.raspberrypi.org/about/

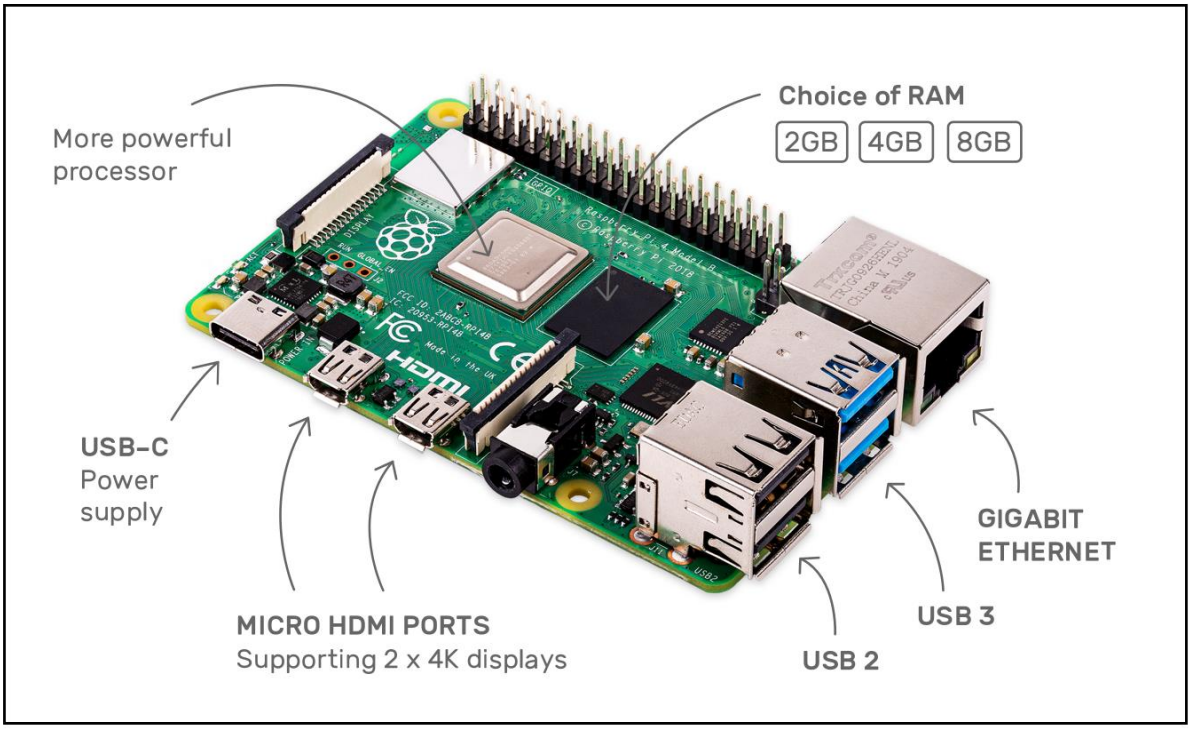


Figura 8- Il Raspberry PI 4

Un altro aspetto interessante, che ha contribuito in maniera fondamentale alla sua diffusione, deriva dal fatto che il Raspberry PI può comunicare facilmente con il mondo esterno grazie ai suoi Pin GPIO (General Purpose Input Output) disponibili direttamente sulla scheda.

Ciò fa si che la scheda possa comunicare facilmente con sensori esterni, LED, interruttori a relé, motori e tante altre componenti elettriche o elettroniche.

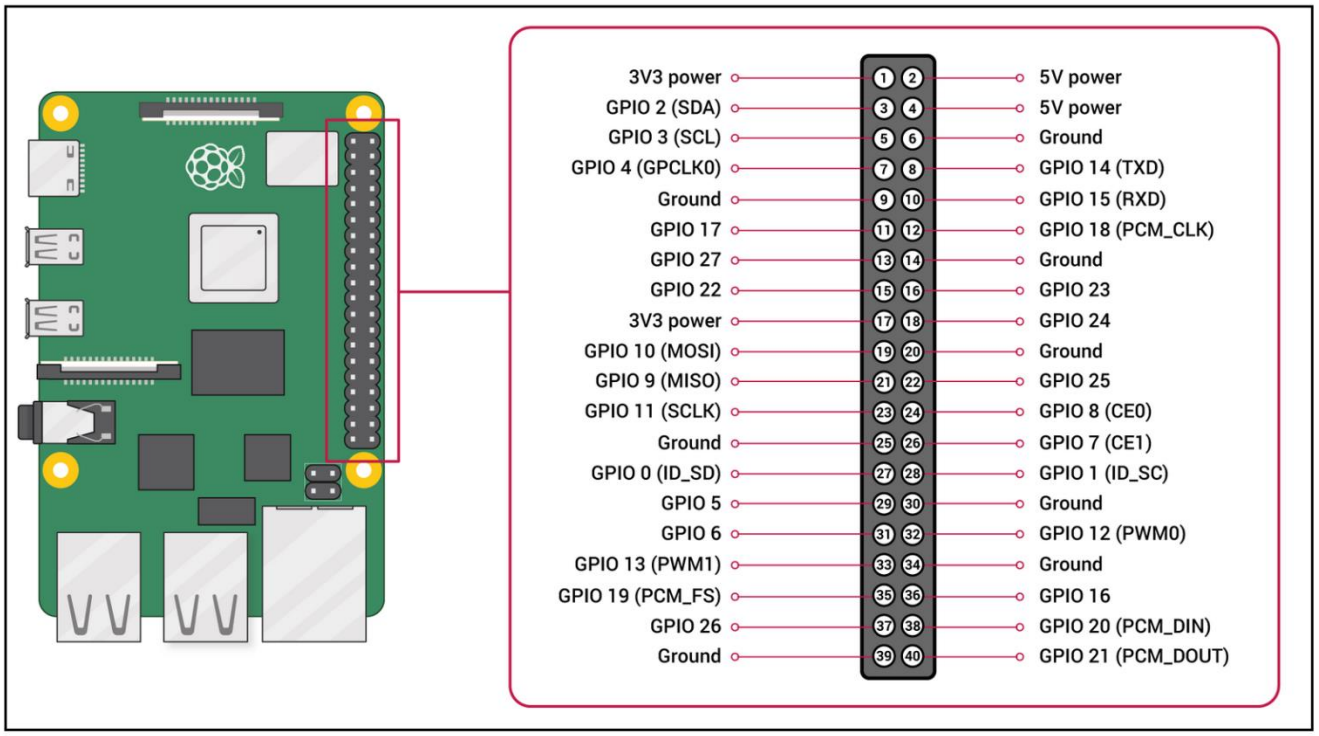


Figura 9- Piedinatura del GPIO del Raspberry PI 4

Tutto ciò rende il Raspberry PI uno dei dispositivi più utilizzati per le applicazioni in ambito IOT.

4.1.1 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.

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 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 quindi il Rapsberry PI, si attiverà una procedura guidata che eseguirà l'installazone del Sistema Operativo Raspbian sul Raspberry PI.

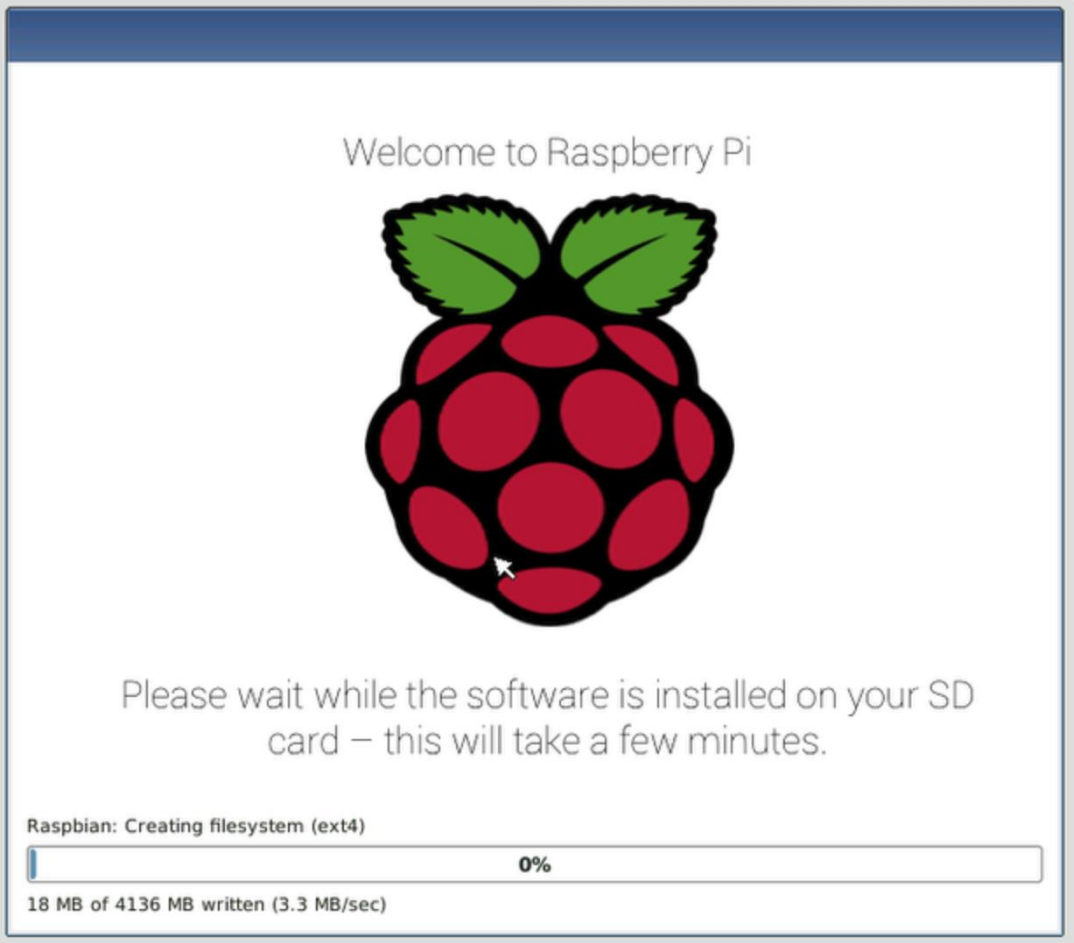


Figura 10- Il processo di installazione di Raspbian OS

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

https://projects.raspberrypi.org/en/projects/noobs-install

4.1.2 Installazione del driver PCAN USB su Raspberry PI

Per installare il driver PCAN sul Raspberry PI 4B, occorre seguire i passi riportati sulla documentazione del sito della Peak System, al seguente link:

https://www.peak-system.com/fileadmin/media/linux/files/PCAN-Driver-Linux_UserMan_eng.pdf.

Al contrario del 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.

Occorre quindi innanzitutto eseguire il download del pacchetto seguendo l'apposito link disponibile alla pagina:

https://www.peak-system.com/fileadmin/media/linux/index.htm

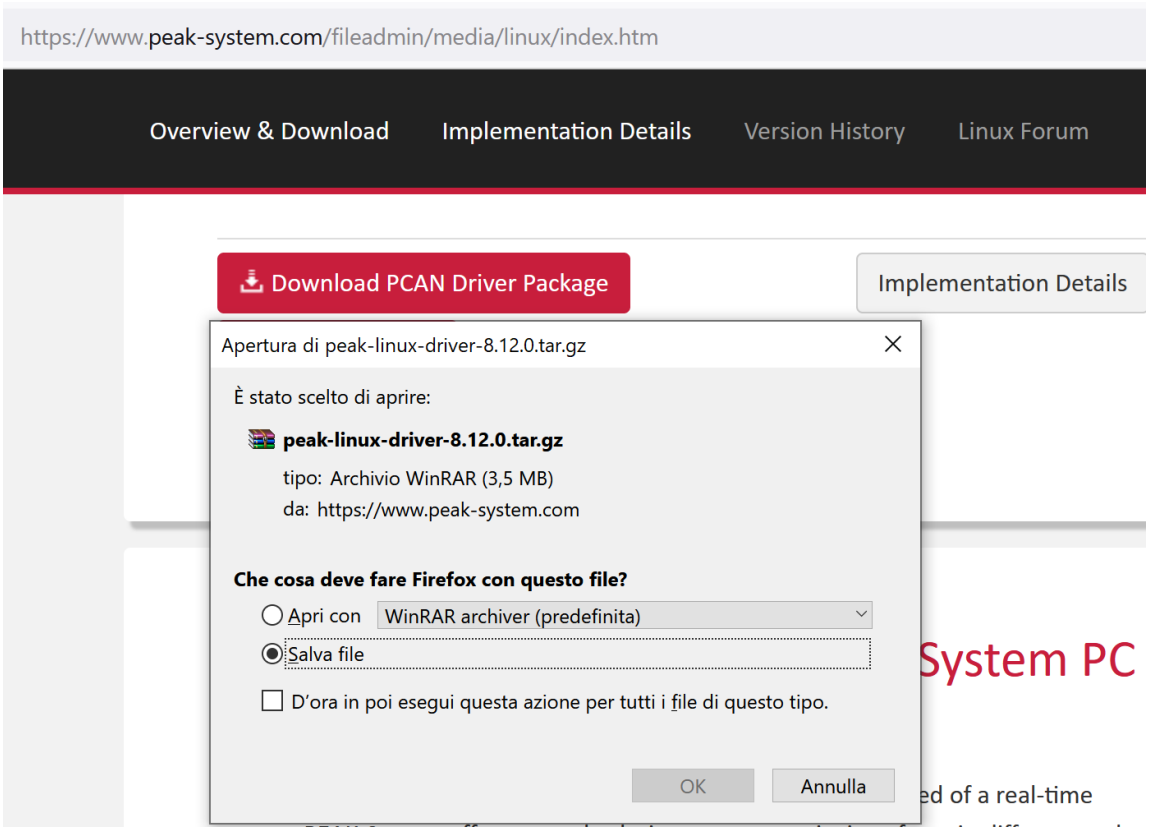


Figura 11- Download del driver del dispositivo PCAN USB

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***

E' 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:

- Il comando ***> dmesg | grep pcan*** elenca tutte le interfacce PCAN, ciasucna 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 12- 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.

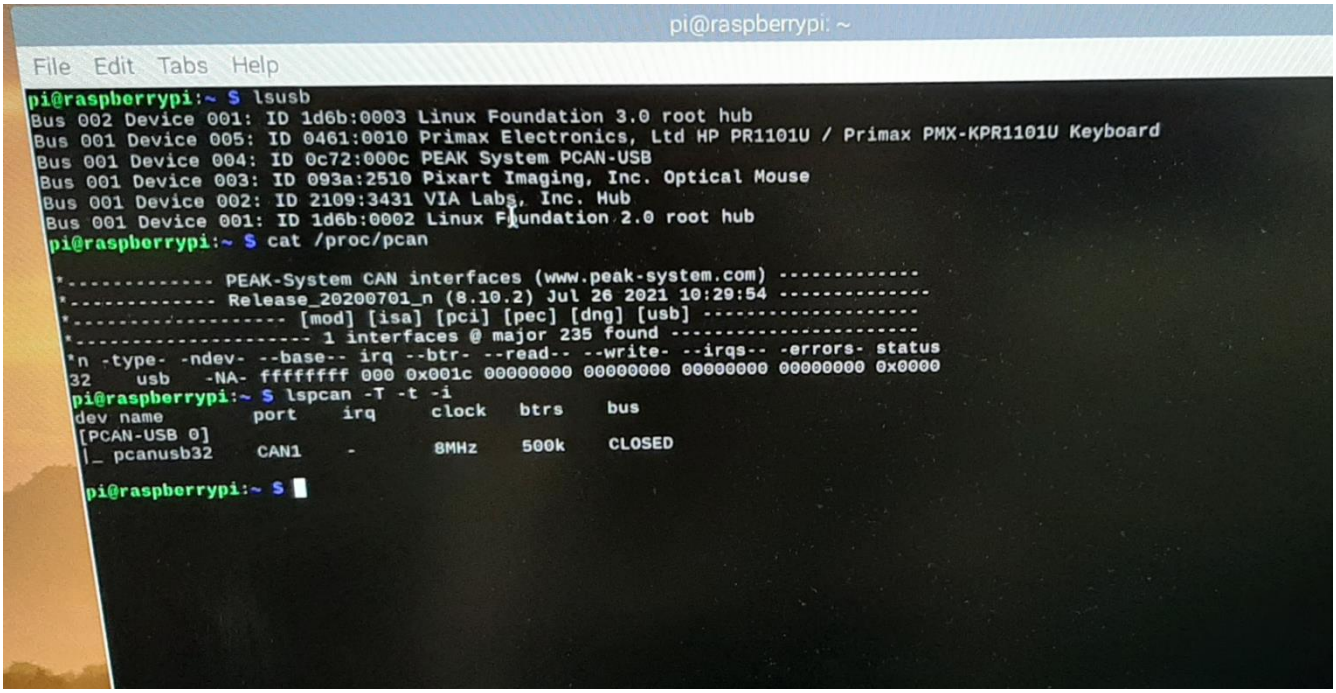


Figura 13- I comandi di verifica del driver PCAN USB

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

4.2 Acquisizione dei dati dal veicolo

Il Raspberry PI 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, ma bisogna tenere conto che il limite fisico imposto dai mezzi utilizzati, non consente di ridurre l'intervallo di rilevazione al di sotto dei 60 secondi.

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.

E' 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 occorre utilizzare un alimentatore per auto che fornisca in uscita almeno la corrente di 3A.

Questa scelta obbligata impatta sicuramente sulle prestazioni della batteria del veicolo, per quanto riguarda la durata della carica, però è un compromesso necessario al funzionamento del sistema di monitoraggio del veicolo. Il modem invece ha una batteria ricaricabile, la cui durata in modalità operativa è di circa 8h, per cui non impatta in maniera significativa sull'efficienza energetica del veicolo.

4.2.1 Il software di acquisizione dati dal veicolo

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.

E' vero però che la maggior parte di queste librerie sono sviluppate facendo uso di linguaggi e tecnologie tipiche dei sistemi Microsoft, e ciò rende obbligatorio l'utilizzo di dispositivi compatibili con il Sistema Operativo Windows 10 IoT[7], sviluppato appositamente da Microsoft per l'utilizzo in ambito IoT.

Il limite legato all'utilizzo di tali sistemi, deriva dal fatto che Windows 10 IoT, è rilasciato in due differenti versioni. La prima distribuita con licenza gratuita, possiede però funzionalità molto limitate e caratteristiche e prestazioni minime, che non lo rendono adatto ad un utilizzo per lo sviluppo di applicazioni professionali avanzate.

Questa versione inoltre, è installabile su dispositivi Raspberry PI, ma solo su quelli di categoria 3 (Raspberry PI 3), che dispongono di un processore fino a 1GHZ e di soltanto 1GB di RAM. Le prove effettuate su questa classe di dispositivi (Raspberry PI 3 con Sistema Operativo Windows 10 IoT), hanno infatti generato un sistema molto instabile e poco performante, praticamente inutilizzabile per i nostri scopi.

La versione avanzata di Windows 10 IoT invece, che è più performante, presenta costi di licenza molto elevati.

Questa considerazione ci ha portato a convergere verso una soluzione totalmente Linux Oriented, prendendo in considerazione soltanto sistemi e dispositivi tipici del mondo Linux.

Raspberry PI 4 quindi, ma anche software e librerie utilizzabili su sistema operativo Linux.

In quest'ottica, è stato necessario scegliere un dispositivo CAN-USB che disponesse di librerie software scritte in linguaggio C, che è il linguaggio di riferimento dei sistemi operativi basati su Linux.

All'Appendice A è riportato il codice del software in linguaggio C. Alla libreria predefinita del dispositivo CAN-USB, che consente di leggere i codici OBD inviati sul CAN Bus, abbiamo aggiunto diverse funzioni, in grado di decodificare i frame OBD ed estrarre da questi le informazioni desiderate. La funzione main del software

[7] https://docs.microsoft.com/it-it/windows/iot-core/windows-iot

esegue un loop durante il quale tutti i frame OBD inviati sul CAN Bus, vengono analizzati e a seconda del codice identificativo letto, viene richiamata la funzione designata a leggere l'informazione contenuta all'interno del frame. Tutte le informazioni così acquisite vengono archiviate su un file locale.
Informazioni rilevate di particolare interesse sono ad esempio lo stato di carica della batteria (SoC) e la velocità del veicolo.

4.3 Acquisizione delle coordinate spaziali

Al fine di acquisire la posizione del veicolo in movimento è stato utilizzato un rilevatore GPS collegato alla porta USB del Raspberry PI. Affinché sia possibile rilevare la posizione istantanea del dispositivo, occorre a questo punto installare il servizio ***gpsd*** sul Raspberry PI.
Il comando da utilizzare è il seguente: ***sudo apt-get install gpsd gpsd-clients***.

Dopo l'installazione il servizio è già attivo, e dopo aver collegato il ricevitore GPS alla porta USB, è possibile leggere i messaggi istantanei inviati dal ricevitore alla seriale utilizzando semplicemente il seguente comando : ***cat /dev/serial0***

Dove *"/dev/serial0"* è il nome (fittizio in questo esempio) dell'interfaccia socket che il sistema ha creato per comunicare con il ricevitore GPS.

Non occorre quindi installare alcun software driver, il Raspberry PI rileva il dispositivo inserito e crea una interfaccia socket seriale per comunicare con esso, ma occorre comunque individuare il nome reale che il sistema dà a tale interfaccia. A tal fine ci soccorrono alcuni comandi del sistema operativo Linux.
Innanzitutto è utile utilizzare il comando ***lsusb***:

```
pi@raspberrypi:~ $
pi@raspberrypi:~ $ lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 005: ID 0000:3825
Bus 001 Device 004: ID 04d9:0006 Holtek Semiconductor, Inc.
Bus 001 Device 003: ID 05e3:0610 Genesys Logic, Inc. 4-port hub
Bus 001 Device 002: ID 2109:3431 VIA Labs, Inc. Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
pi@raspberrypi:~ $
pi@raspberrypi:~ $
pi@raspberrypi:~ $ lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 005: ID 0000:3825
Bus 001 Device 004: ID 04d9:0006 Holtek Semiconductor, Inc.
Bus 001 Device 003: ID 05e3:0610 Genesys Logic, Inc. 4-port hub
Bus 001 Device 006: ID 1546:01a8 U-Blox AG [u-blox 8]
Bus 001 Device 002: ID 2109:3431 VIA Labs, Inc. Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
pi@raspberrypi:~ $
```

Figura 14- Il comando lsusb

Il comando ***lsusb*** elenca tutti i dispositivi collegati alle prese USB del computer, e ci permette di verificare la situazione del sistema, prima e dopo l'inserimento del dispositivo GPS nella presa USB.

Come si vede dalla figura, il dispositivo GPS viene rilevato sul Bus 001 come dispositivo U-Blox AG (modulo GPS Glonass Vk-172).

Il comando ***dmesg*** invece, ci restituisce il dettaglio delle caratteristiche del dispositivo inserito. Nel nostro caso, come si vede in figura, viene rilevato un nuovo dispositivo USB, viene elencato il tipo di dispositivo ed il produttore, e viene generato un nuovo device socket ttyACM0 per gestirlo.

```
pi@raspberrypi:~ $
pi@raspberrypi:~ $
pi@raspberrypi:~ $ dmesg
[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Linux version 5.10.17-v7l+ (dom@buildbot) (arm-linux-gnueabihf-gcc-8 (Ubuntu/Linaro 8.4.0-3ubuntu1) 8.4.0, GNU ld (GNU Binutils for Ubuntu) 2.34) #142
1 SMP Thu May 27 14:00:13 BST 2021
[    0.000000] CPU: ARMv7 Processor [410fd083] revision 3 (ARMv7), cr=30c5383d
[    0.000000] CPU: div instructions available: patching division code
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, PIPT instruction cache
[    0.000000] OF: fdt: Machine model: Raspberry Pi 4 Model B Rev 1.4
[    0.000000] Memory policy: Data cache writealloc
[    0.000000] Reserved memory: created CMA memory pool at 0x000000001ec00000, size 256 MiB
[    0.000000] OF: reserved mem: initialized node linux,cma, compatible id shared-dma-pool
[    0.000000] Zone ranges:
[    0.000000]   DMA      [mem 0x0000000000000000-0x000000002fffffff]
[    0.000000]   Normal   empty
[    0.000000]   HighMem  [mem 0x0000000030000000-0x00000001ffffffff]
[    0.000000] Movable zone start for each node
[    0.000000] Early memory node ranges
[    0.000000]   node   0: [mem 0x0000000000000000-0x000000003b3fffff]
[    0.000000]   node   0: [mem 0x0000000040000000-0x00000000fbffffff]
[    0.000000]   node   0: [mem 0x0000000100000000-0x00000001ffffffff]
[    0.000000] Initmem setup node 0 [mem 0x0000000000000000-0x00000001ffffffff]
[    0.000000] On node 0 totalpages: 2061312
[    0.000000]   DMA zone: 2304 pages used for memmap
[    0.000000]   DMA zone: 0 pages reserved
[    0.000000]   DMA zone: 196608 pages, LIFO batch:63
[    0.000000]   HighMem zone: 1864704 pages, LIFO batch:63
[    0.000000] percpu: Embedded 20 pages/cpu s50700 r8192 d23028 u81920
[    0.000000] pcpu-alloc: s50700 r8192 d23028 u81920 alloc=20*4096
[    0.000000] pcpu-alloc: [0] 0 [0] 1 [0] 2 [0] 3
[    0.000000] Built 1 zonelists, mobility grouping on.  Total pages: 2059008
[    0.000000] Kernel command line: coherent_pool=1M 8250.nr_uarts=0 snd_bcm2835.enable_compat_alsa=0 snd_bcm2835.enable_hdmi=1 video=HDMI-A-1:1920x1080M@60 smsc95xx
.macaddr=DC:A6:32:D7:06:04 vc_mem.mem_base=0x3ec00000 vc_mem.mem_size=0x40000000  console=ttyS0,115200 console=tty1 root=/dev/mmcblk0p7 rootfstype=ext4 elevator=dead
line fsck.repair=yes rootwait quiet splash plymouth.ignore-serial-consoles
[    0.000000] Kernel parameter elevator= does not have any effect anymore.
                Please use sysfs to set IO scheduler for individual devices.
[    0.000000] Dentry cache hash table entries: 131072 (order: 7, 524288 bytes, linear)
[    0.000000] Inode-cache hash table entries: 65536 (order: 6, 262144 bytes, linear)
```

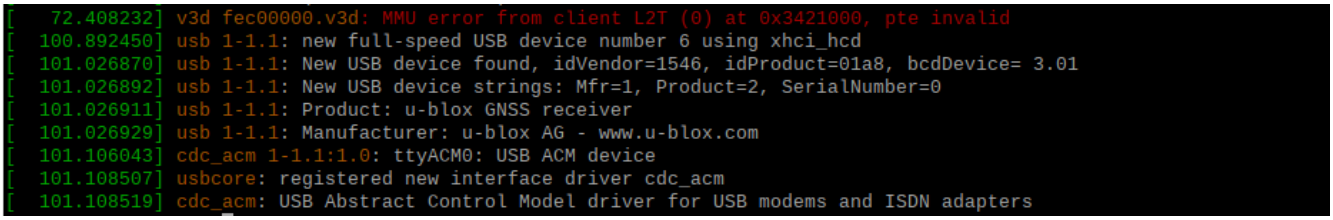


Figura 15- Il comando dmesg

Il nostro dispositivo di riferimento è quindi **/dev/ttyACM0** e quindi il comando da eseguire per visualizzare a schermo tutti i messaggi del ricevitore è il seguente: ***cat /dev/ttyACM0***

Eseguendo questo comando si visualizzano in tempo reale i messaggi inviati dal ricevitore GPS sulla seriale:

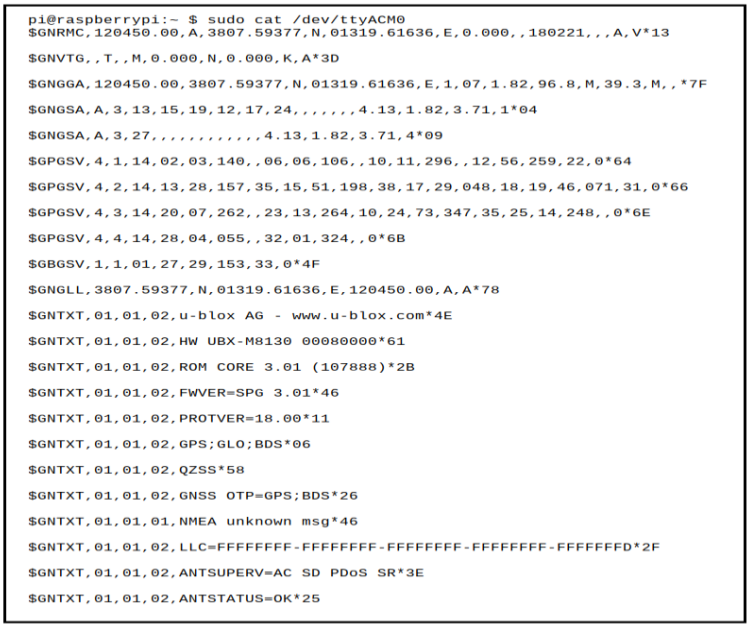


Figura 16- Messaggi del ricevitore GPS

Si tratta di stringhe di caratteri scritte secondo lo standard NMEA 0183 (National Marine Electronics Association[8]). Secondo tale standard, ogni stringa di caratteri è formata secondo una regola precisa e assume un altrettanto preciso contenuto informativo:

[8] https://www.nmea.org/content/standards/nmea_0183_standard

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)

Checksum range

Start of Sequence	Address field	Data Field(s)	Checksum field	End of Sequence
$	<Address>	[,<data field>].....[,<data field>]	*<checksum>	<CR><LF>

Figura 17- 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" 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 18- Address fields NMEA

4.3.1 Lo script "readSerial.py"

Al fine di automatizzare il processo di lettura e codifica dei messaggi inviati dal ricevitore GPS, è stato messo a punto uno script in linguaggio Python[9], 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.

[9] https://www.python.org

```
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 19- 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.

4.4 Sincronizzazione degli eventi temporali: il demone crond di Linux

Il demone ***crond*** è il servizio che consente la schedulazione dell'esecuzione dei processi in ambiente Linux. Il demone è sempre in esecuzione, ed ogni minuto controlla il suo registro dei processi da avviare, chiamato ***crontab***. Il servizio, legge il crontab[10] ed avvia l'esecuzione dei processi che sono stati schedulati per quel preciso istante temporale.

Per schedulare l'esecuzione di uno o più processi è quindi necessario inserirli in forma corretta all'interno del registro crontab. Ciò può essere effettuato eseguendo il seguente comando: ***crontab –e***

Per editare il crontab bisogna seguire la sintassi corretta, che prevede innanzitutto l'indicazione del minuto di avvio del processo, poi bisogna indicare l'ora, poi il giorno del mese, quindi il mese, poi il giorno della settimana. Infine, sempre sulla stessa riga, occorre indicare il comando da eseguire specificando il path completo ed un

[10] https://manpages.debian.org/buster/cron/crontab.5.en.html

eventale file di log per registrare il risultato dell'esecuzione.

Nella figura seguente è indicata la corretta sintassi del crontab.

```
# .---------------- [m]inute: minuto (0 - 59)
# |  .------------- [h]our: ora (0 - 23)
# |  |  .---------- [d]ay of month: giorno del mese (1 - 31)
# |  |  |  .------- [mon]th: mese (1 - 12) OPPURE jan,feb,mar,apr...
# |  |  |  |  .---- [w]eek day: giorno della settimana (0 - 6) (domenica=0 o 7)
# |  |  |  |  |                 OPPURE sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  *  comando da eseguire
```

Figura 20- La sintassi del crontab

Come si vede dalla figura, per tutti i valori temporali, è possibile indicare un asterisco al posto di un valore numerico, per indicare tutti i valori ammessi.

E' anche possibile specificare più valori numerici separati dalla virgola, per indicare che l'esecuzione deve avvenire solo ed esclusivamente nell'istante specificato, oppure indicare un intervallo di valori specificando solo i due valori agli estremi dell'intervallo separati da un trattino.

Per visualizzare invece l'elenco dei processi schedulati, basta digitare il seguente comando : ***crontab -l***

Nel nostro sistema, tale comando mostra la situazione illustrata nell'immagine seguente:

```
pi@raspberrypi:~ $ crontab -l
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
#15 10,11,12,14 * * * /home/pi/PCAN/PCAN-Basic_Linux-4.4.1/libpcanbasic/examples/c++/myNissanRead >> /home/pi/PCAN/cronlog.txt 2>&1
#15 10,11,12,14 * * * python /home/pi/readSerial.py >> /home/pi/PCAN/cronlog.txt 2>&1
#16 10,11,12,14 * * * python /home/pi/uploadftp.py gpsData.csv >> /home/pi/PCAN/cronlog.txt 2>&1
#17 10,11,12,14 * * * python /home/pi/uploadftp.py datalog.txt >> /home/pi/PCAN/cronlog.txt 2>&1
pi@raspberrypi:~ $
```

Figura 21- Il crontab

Sono quindi schedulati quattro processi che verranno avviati lanciando in esecuzione i seguenti script :

1. *myNissanRead*
2. *readSerial*
3. *uploadftp gpsData.csv*

4. *uploadftp datalog.txt*

Come si vede nell'esempio, i processi saranno avviati esattamente al minuto 15 delle ore 10,11,12 e 14 di tutti i giorni del mese, di tutti i mesi, e in tutti i giorni della settimana.
Notiamo anche che nel crontab occorre indicare il path completo per l'esecuzione dello script.

4.5 I protocolli di trasferimento dati SSH e FTP

Per il trasferimento dei dati rilevati verso il server di destinazione, ci si è serviti dei protocolli di comunicazione FTP e SSH.
Il protocollo FTP (File Transfer Protocol) è un protocollo del livello applicazioni della pila di riferimento del modello ISO/OSI. E' molto usato per il download o per l'upload di file da un client ad un server e viceversa. Occorre quindi una macchina server che metta a disposizione il servizio FTP che verrà poi utilizzato da una macchina client. E' richiesta l'autenticazione del client sul server, si tratta però di un protocollo poco adatto alle connessioni che necessitano di una certa sicurezza, perché, nella versione base del protocollo, sia la password di autenticazione che le ulteriori comunicazioni, sono tutte trasmesse in chiaro. Per questo motivo è spesso usato nella sua versione denominata FTPS, in cui viene aggiunto al protocollo originario un layer di cifratura SSL/TLS.
Si tratta inoltre di un protocollo orientato alla sessione. Il client che fa richiesta di connessione al server può restare inattivo solo per un tempo limitato (idle timeout) trascorso il quale la sessione verrà comunque chiusa e terminata dal server.
Il protocollo SSH [12] invece è il protocollo di riferimento per le connessioni sicure, perché è in grado di stabilire una sessione cifrata tra client e server. E' 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.
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 seguente 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***

```
pi@raspberrypi:~ $
pi@raspberrypi:~ $
pi@raspberrypi:~ $ ssh-keygen -t rsa -b 1024
Generating public/private rsa key pair.
Enter file in which to save the key (/home/pi/.ssh/id_rsa): id_rsa2
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in id_rsa2.
Your public key has been saved in id_rsa2.pub.
The key fingerprint is:
SHA256:YXn5MWuVPRIowfjBD1ce0F8xz2VKXJuU3ToPKjMapCY pi@raspberrypi
The key's randomart image is:
+---[RSA 1024]----+
|        +...==.=O|
|       . B +.o=BX|
|        = O oo===|
|       ..+ o ==..|
|       oS   +. + |
|    E o . +..   .|
|     o   o +     |
|        .        |
|                 |
+----[SHA256]-----+
pi@raspberrypi:~ $ ssh-copy-id -i id_rsa2.pub giuseppe@apic.casaccia.enea.it
```

Figura 22- Generazione di una coppia di chiavi pubblica e privata

4.5.1 Lo script "uploadftp.py"

```
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 23- Lo script di trasferimento dati

Lo script *uploadftp.py* 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.

4.6 Lo script di connessione VPN

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 tramissione 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.

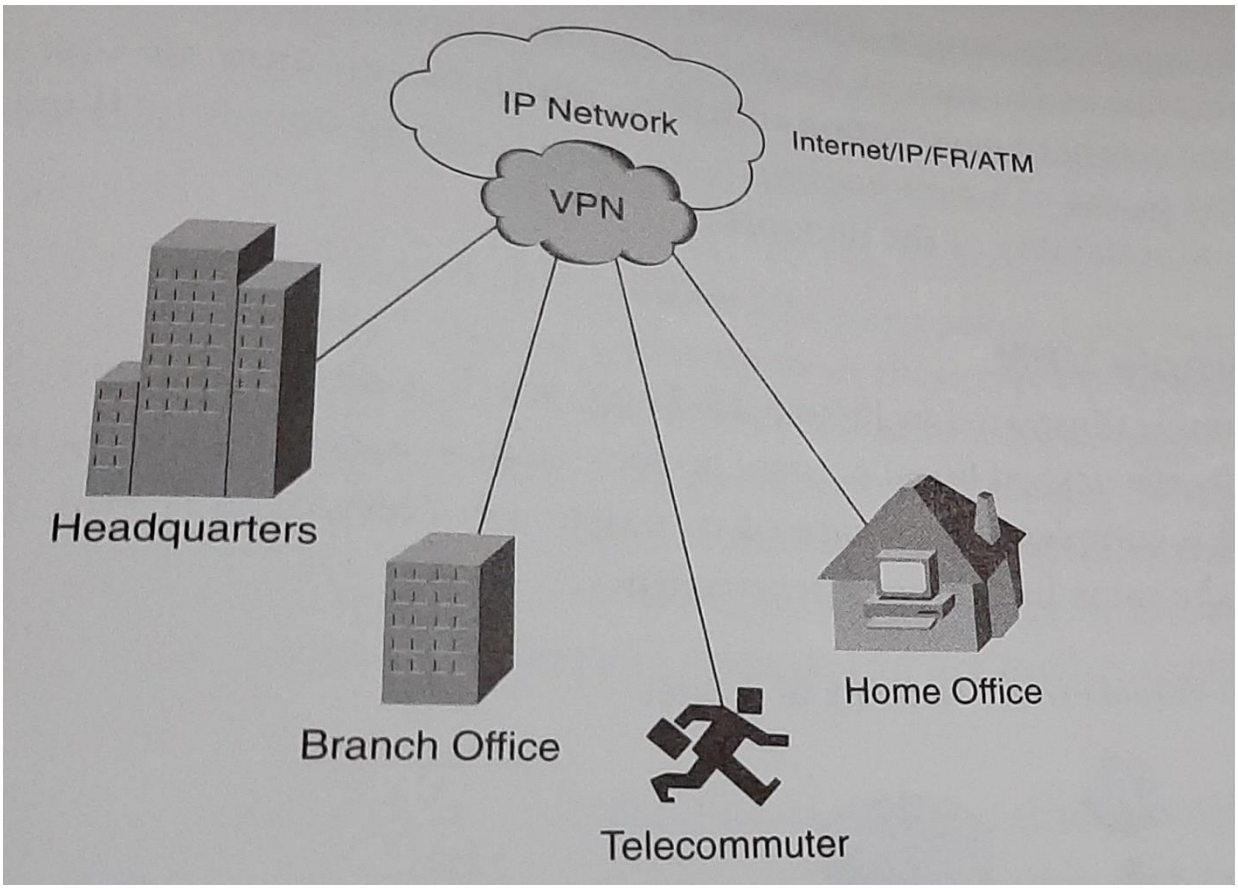


Figura 24- Schema di una VPN

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 seguente 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 25- Lo script di connessione alla rete VPN

5. Acquisizione delle informazioni dal territorio

Per quanto riguarda invece i sensori utilizzati per rilevare le informazioni dal territorio, negli ultimi anni sono stati sviluppati molti studi e ricerche nel campo delle Smart Cities, ed in particolare dei sistemi di monitoraggio degli spazi di sosta all'interno di un parcheggio (Vehicle Detection System). Tali sistemi si basano tutti sulla rilevazione delle informazioni da opportuni sensori dislocati in posizione strategica sul territorio, in modo da rilevare le informazioni richieste e trasferirle ad un server remoto per le opportune elaborazioni.

I sensori più utilizzati per questi scopi sono di due tipi:

1. ***Sensori magnetici*** da installare al suolo [13]. E' necessaria la presenza di un sensore per ciascun posto auto da monitorare. E' possibile l'installazione sulla superficie dell'asfalto, in modo da evitare operazioni di scavo, oppure sotto l'asfalto. La durata della batteria del sensore varia tra 8 e 12 anni, a seconda del modello di sensore utilizzato, ed è sostituibile accedendo all'apposito alloggiamento dislocato sulla superficie del sensore. I sensori sono dotati di connettività wireless per la comunicazione tramite onde radio con un gateway locale. Quest'ultimo si occuperà poi dell'invio delle informazioni al server remoto.

   I protocolli di comunicazione utilizzati a tale scopo sono tutti proprietari, messi a punto dalla casa costruttrice dei sensori stessi. Tra tutti, negli ultimi anni si è affermata la tecnologia denominata LoRa [14], di proprietà dell'azienda Semtech, che permette di effettuare una comunicazione a basso consumo energetico, al fine di preservare il più possibile la carica della batteria dei sensori.
2. ***Sensori ottici*** *(telecamere)* [15] su opportuni pali di sostegno. Il sistema prevede l'utilizzo di una o più telecamere, per l'acquisizione e la successiva elaborazione, dell'immagine istantanea dell'area di

parcheggio da monitorare. L'immagine acquisita deve essere inviata ad un server sul quale è in esecuzione il software di elaborazione, in grado di interpretare l'immagine ed estrarre da essa tutte le informazioni richieste. Queste informazioni andranno poi inviate nel formato idoneo alla piattaforma remota di controllo.

5.1 I sensori magnetici

Per motivi di facilità di realizzazione e di utilizzo, oltre che di economicità, si è preferito in questa fase, optare per la prima soluzione. Nella figura seguente lo schema rappresentativo della soluzione con sensori magnetici.

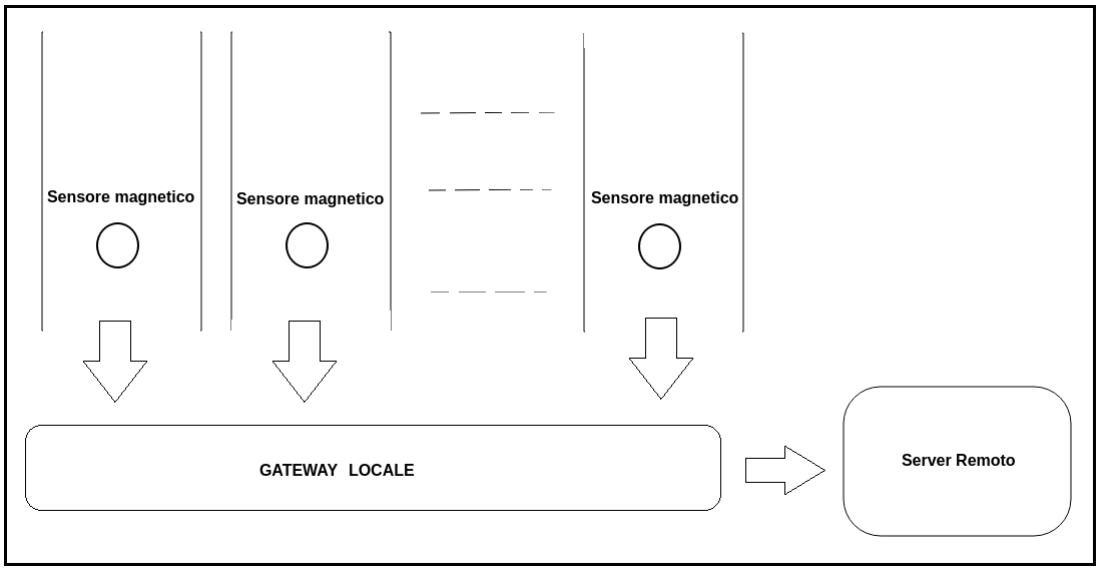


Figura 26- Soluzione con i sensori magnetici

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 al gateway 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 *Gateway Locale* 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 seguente un esempio di record ottenuto in rilevazione:

codice ID	latitudine	longitudine	data e ora	busy
0001	38.1319879	13.3240785	2021-07-16 11:05:32	True/False

Figura 27- Il record delle informazioni rilevate

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

6. Conclusioni

Il sistema completo è stato provato con successo, riportiamo in questa sezione gli output e i risultati ottenuti.

Per attivare il sistema basta 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. Come si vede dalla Fig.21, 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 files 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 seguente mostra un esempio del file "*gpsData.csv"*:

```
1;2021-07-22 14:25:02;865;38.1265616667;13.3269383333
```

Figura 28- 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"* contiene invece i dati rilevati dal veicolo.

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 29- I dati rilevati dal veicolo

7. Riferimenti

1. C. Liberto, G. Valenti, "*Simulatore di scenari urbani di mobilità veicolare elettrica*", Rapporto RdS/PAR2017/241, 2018
2. M. Pollino, L. La Porta, A. Di Pietro, A. Tofani, E. Caiaffa, V. Rosato, "*La piattaforma per la sicurezza delle Infrastrutture Critiche*", Rapporto RdS/PAR2015/016
3. M. Bartolozzi, P. Bellini, P. Nesi, G. Pantaleo, L. Santi. "*A Smart Decision Support System for Smart City*". 2015 IEEE International Conference on Smart City/SocialCom/SustainCom together with DataCom 2015 and SC2 2015 At: Chengdu, Sichuan, China
4. S. Allam And U. Elhady. *"On the Development and Implementation of the OBD II Vehicle Diagnosis System". International Journal of Engineering Inventions e-ISSN : 2278-7461, p-ISSN : 2319-6491*

Volume 7, Issue 4 [April 2018] PP: 19-27

5. *CSS Electronics, "OBD-II Explained a Simple Intro (2021)", https://www.csselectronics.com/pages/obd2-explained-simple-intro*
6. BOSCH, Robert. 2007 Bosch Automotive Automotive Electrics and Electronics. Plochingen: Robert Bosch GmbH. 2007. 500p. ISBN 978-3-658-01783-5.
7. CAN Specification 2.0, Part A and Part B. http://www.can-cia.org/
8. CSS Electronics, "*CAN Bus Explained, a Simple Intro(2021)*", https://www.csselectronics.com/pages/can-bus-simple-intro-tutorial
9. Ortenzi, F. and Costagliola, M., *"A New Method to Calculate Instantaneous Vehicle Emissions using OBD Data"* SAE Technical Paper 2010-01-1289, 2010, https://doi.org/10.4271/2010-01-1289
10. A. Alessandrini, F. Filippi, F. Ortenzi. "*Consumption calculation of vehicles using OBD data*", 20th International Emission Inventory Conference, August 2012
11. K.M. Abdul Kadhar and G.Arnand, "*Data Science with Raspberry PI*", Springer 2021, https://doi.org/10.1007/978-1-4842-6825-4_3
12. Stephen C. Williams, *"Analysis of the SSH Key Exchange Protocol"*, 13th IMA International Conference, IMACC 2011, Oxford, UK, December 12-15, 2011. Proceedings
13. M. BUGDOL, Z. SEGIET, M. KRĘCICHWOST. "*Vehicle Detection System Using Magnetic Sensors*". March 2014 Transport Problems 9(1):49-60
14. Jesus Sanchez-Gomez e Ramon Sanchez-Iborra, *"Experimental comparison of LoRa and FSK as IoT-communication-enabling modulations"*, in *IEEE Global Communications Conference (Globecom'17)*, 2017, pp.1–6
15. H. Chandel, S. Vatta. "*A Vision based Vehicle Detection System*". Communications on Applied Electronics (CAE) – ISSN : 2394-4714 Foundation of Computer Science FCS, New York, USA Volume 2 – No.6, August 2015 – www.caeaccess.org

8. Abbreviazioni e acronimi

IOT	Internet Of Thinghs
OBD	On Board Diagnostic system
CARB	California Air Resources Board

EPA	Environmental Protection Agency
PID	Parameter IDentifier
DTC	Diagnostic Trouble Codes
CAN	Controller Area Network
ECU	Electronic Control Unit
API	Application Program Interface
GSM	Global System for Mobile communications
GPS	Global Positioning System
SBC	Single Board Computer
NMEA	National Marine Electronics Association
SSH	Secure SHell
FTP	File Transfer Protocol
LAN	Local Area Network
VPN	Virtual Private Network

Appendice A. Il codice del software di acquisizione dati

```
#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;
}
```

Appendice B. Elenco delle figure