

UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI INGEGNERIA

Corso di Laurea triennale in Ingegneria Informatica e dell'Automazione

Formula SAE: progettazione e sviluppo del software per ottimizzare le prestazioni e l'affidabilità di un veicolo elettrico.

Formula SAE: design and development of software to optimize the performance and reliability of an electric vehicle.

Relatore: Chiar.mo/a	Correlatore: Chiar.mo/a	Tesi di laurea di:
Prof. Ippoliti Gianluca	**Prof. Orlando Giuseppe**	**Giuliani Matteo**

A.A. 2022/2023

Introduzione

Questo lavoro si concentra sulla progettazione e lo sviluppo del software che gestisce una parte del veicolo elettrico. Nella parte iniziale della tesi si parla della storia della Formula SAE e come sono strutturate le gare. Viene poi introdotta la storia della squadra che rappresenta l'università Politecnica delle Marche, il Polimarche Racing Team, e di come si è evoluta negli anni.

In seguito, viene spiegata l'architettura elettrica del veicolo, di quali parti si compone e di quali sensori è dotato. Successivamente viene dettagliato il funzionamento del PDM (Power Distribution Module) all'interno del veicolo insieme alla spiegazione del software che risiede sulla scheda di acquisizione, riportando anche il codice C++ completo. In riferimento a tale argomento, si fa chiarezza sulla storia del CAN BUS, a cosa serve e come è stato implementato all'interno del veicolo.

Infine, viene spiegato il funzionamento del GPS e in che modo risulta essere utile nel contesto di una gara automobilistica, riportando la strategia software adottata per il nostro veicolo elettrico.

Indice dei capitoli

CAPITOLO 1 FORMULA SAE E POLIMARCHE 6

1.1 Struttura di una gara 6

1.2 Polimarche 8

CAPITOLO 2 INTRODUZIONE ELETTRICA DEL VEICOLO 12

2.1 Architettura del sistema elettrico 13

2.2 Logica del layout elettrico 14

2.3 Sensori 16

2.4 Schede di acquisizione e ambiente di sviluppo 17

CAPITOLO 3 SOFTWARE PDM 18

3.1 Acquisizione dati 20

3.1.1 Acquisizione delle temperature 20

3.1.2 Acquisizione delle correnti 21

3.1.3 Acquisizione delle tensioni 22

3.1.4 Controllo dei relè 23

3.1.5 Generazione segnali PWM e segnali luce dei freni 25

3.1.6 Librerie C++ 27

Temperature.h 27
Temperature.cpp 27
Currents.h 28
Currents.cpp 29
Voltages.h 30
Voltages.cpp 30
PWMHandler.h 31
PWMHandler.cpp 31
RelayHandler.h 32
RelayHandler.cpp 32

3.2 Comunicazione CANBUS 33

3.2.1 Funzionamento del CAN 33

3.2.2 Progettazione del software 35

3.2.3 Librerie C++ 38

Canbus.h 38
Canbus.cpp 39

3.3 Codice completo del software PDM 42

CAPITOLO 4 TRACKING GPS 43
4.1 Funzionamento del GPS 43
4.2 Progettazione del software GPS 44
4.3 Testing su MATLAB 46
4.3.1 Codice MATLAB 47
GPS_Google_Earth.m 47
GPS_Tracking.m 48
4.4 Codice completo del software GPS 49
CAPITOLO 5 SITOGRAFIA 53
CAPITOLO 6 RINGRAZIAMENTI 54

Indice delle figure

Figura 1-1. Layout dell'evento Skid Pad 7
Figura 1-2. Sviluppo della macchina negli anni 8
Figura 1-3. Mockup Peacock 1 8
Figura 1-4. Mockup Peacock 2 9
Figura 1-5. Mockup Peacock 3 9
Figura 1-6. Mockup Peacock 3 EVO 10
Figura 1-7. Mockup Peacock 4 10
Figura 1-8. Mockup della Peacock 5 11
Figura 2-1. Architettura elettrica generale 13
Figura 2-2. Architettura elettrica del veicolo 15
Figura 2-3. Lista dei sensori 16
Figura 3-1. Schema logico PDM 18
Figura 3-2. Esempio di gestione dei dati del PDM 19
Figura 3-3. Sensori di temperatura 20
Figura 3-4. Correnti sull’ADC esterno 21
Figura 3-5. Sensori di tensione 22
Figura 3-6. Schema dei relè 23
Figura 3-7. Sensori PWM e luce freni 25
Figura 3-8. Struttura di un messaggio CAN BUS 33
Figura 3-9. Struttura di un messaggio di gestione degli errori 34
Figura 3-10. Suddivisione dei bytes per indirizzo 36
Figura 3-11. Nodo trasmittente, 2 byte per dato 36
Figura 3-12. Nodo ricevente, 2 byte per dato 37
Figura 3-13. Nodo trasmittente, 1 byte per dato 37
Figura 3-14. Nodo ricevente, 1 byte per dato 37
Figura 4-1. Trilaterazione 2D 44
Figura 4-2. Trilaterazione 3D 44
Figura 4-3. Circonferenza di punti attorno alla coordinata iniziale 45
Figura 4-4. Tracciato simulato 46
Figura 4-5. Tracciato coordinate GPS 46

Capitolo 1
FORMULA SAE E POLIMARCHE

Formula SAE è un progetto per studenti che coinvolge i dipartimenti di ingegneria delle Università di tutto il mondo organizzato da SAE International. Il concorso è stato lanciato nel 1980 dal ramo studentesco SAE dell'Università del Texas (Austin). Il concetto alla base della Formula SAE è che una società immaginaria ingaggi un team di studenti affinché realizzino un prototipo di auto da corsa in stile Formula 1. Ogni team di studenti progetta, costruisce e testa un prototipo sulla base di una serie di regole dettate dalla stessa SAE.

1.1 **Struttura di una gara**

La competizione si tiene una volta l'anno ed è suddivisa in 4 categorie ognuna con le sue regole e requisiti specifici:

- Classe 1C (*combusion)*: i veicoli a motore termico.
- Classe 1E (*electric)*: i veicoli a motore elettrico.
- Classe 1D (*driverless)*: i veicoli a guida autonoma.
- Classe 3: dove partecipano i team che presentano solo il progetto della vettura, non partecipano quindi agli eventi statici e dinamici.

Durante la competizione, le squadre affrontano alcune sfide suddivise in due principali eventi, quali eventi statici ed eventi dinamici.
Gli eventi statici sono a loro volta suddivisi in:

- **Design**: L'obiettivo di questo evento è giudicare le scelte e gli sforzi progettuali e come essi possono soddisfare le esigenze di mercato. Viene giudicata anche la capacità di rispondere ai quesiti posti e all'ispezione della macchina.
- **Business**: L'obiettivo di questo evento è quello di valorizzare la capacità dei team nello sviluppo di un *business case* esaustivo.
- **Cost**: L'obiettivo di questo evento è di valutare la capacità degli studenti di pensare in maniera critica le decisioni finanziarie prese. Gli studenti devono essere in grado di valutare i costi e i benefici delle diverse opzioni disponibili e di scegliere la soluzione migliore per il loro progetto.

Invece, gli eventi dinamici sono a loro volta suddivisi in:

- **Acceleration**: Durante il test di accelerazione la vettura deve accelerare per 75 metri lungo un percorso rettilineo su superficie piana. La prova è divisa in due batterie, che devono essere corse da due differenti piloti, ognuno dei quali ha a disposizione due tentativi. Il punteggio è determinato dalla differenza tra il peggiore e il migliore tempo assoluti fatti registrare.

- **Skid Pad**: La prova valuta la capacità in curva della vettura. Il tracciato richiama la figura di un 8 con due cerchi di 15.25 metri, al cui centro è posta la linea di partenza/arrivo. La vettura, una volta entrata nel percorso, deve compiere un giro del cerchio destro, per stabilire il senso di marcia, al termine del quale deve compierne un secondo, il quale viene cronometrato dai giudici. Terminato il secondo giro, la vettura deve spostarsi nel cerchio sinistro per effettuare due ulteriori giri, il secondo dei quali viene cronometrato. La prova è divisa in due batterie, che devono essere corse da due differenti piloti, ognuno dei quali ha a disposizione due tentativi. Il punteggio è determinato sulla capacità di accelerazione laterale.

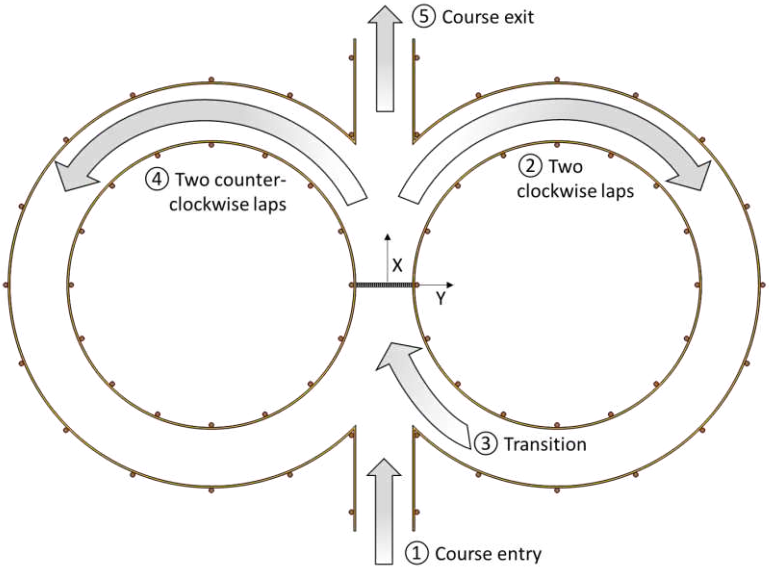


Figura 1-1. Layout dell'evento Skid Pad

- **Autocross**: Prova di sprint da eseguire su due giri di circuito per valutare la maneggevolezza della vettura. Il tracciato è strutturato in modo da ottenere velocità medie comprese tra i 30 ed i 40 km/h e comprende brevi rettilinei, curve a raggio costante, tornanti, slalom tra dei coni e chicanes. Per il punteggio vale il miglior tempo su 4 giri totali divisi tra i due piloti.
- **Endurance**: evento finale che mira a valorizzare l'insieme delle prestazioni in gara di ogni vettura. Si svolge lungo un tracciato molto simile a quello in cui si corre la prova di autocross, per un totale di 22 km. Ai componenti del team non è permesso di intervenire sul veicolo durante lo svolgimento della prova, mentre è previsto un cambio di pilota a metà gara. L'ordine di partenza viene stabilito in base ai risultati dell'autocross, con la squadra più ad aver realizzato il miglior tempo prima a scendere in pista. Il tempo complessivo dell'endurance è dato dalla somma dei tempi di ciascun pilota a cui vanno sommati le eventuali penalità.

1.2 **Polimarche**

Nato nel 2013, il Polimarche Racing Team è la squadra di Formula Student dell'Università Politecnica delle Marche in collaborazione con i vari dipartimenti dell'università che vede coinvolti circa 80 studenti tra i diversi settori dell'ingegneria. Il team è organizzato in maniera gerarchica, infatti si compone di un Team Leader, un Project Manager e diversi capi per ogni reparto (Telaio, Aerodinamica, High Voltage, Low Voltage, Cost) ognuno che dirige i lavori nella propria divisione.
In Figura 1-2 sono mostrate le varie evoluzioni della macchina nei diversi anni. Questa vettura gareggiava nella categoria *combustion* (Classe 1C).

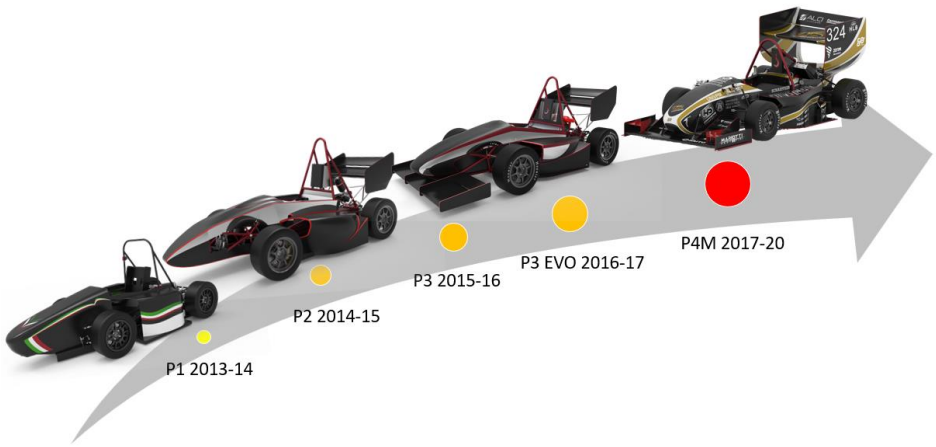


Figura 1-2. Sviluppo della macchina negli anni

Peacock 1: Prima vettura del Polimarche Racing Team, cui progetto e realizzazione è avvenuto in soli 6 mesi. Il risultato più importante tra tutti è stato il completamento di tutti gli eventi dinamici dell'edizione FS Italy 2014 a Varano de Melegari, tra cui anche quello più arduo, l'Endurance Event.



Figura 1-3. Mockup Peacock 1

Peacock 2: Seconda vettura del team, caratterizzata dall'evoluzione del pacchetto aerodinamico, in cui compare per la prima volta l'alettone posteriore. La vettura ha ottenuto buoni risultati nelle gare di Acceleration Event in Formula Czech (3° posto) e nell'Engineering Design Event in Italia nella Formula ATA (4° posto) nel 2015.



Figura 1-4. Mockup Peacock 2

Peacock 3: Terza vettura del team, che vede ancora una volta una miglioria nel pacchetto aerodinamico, andando ad aggiungere agli elementi già presenti (fondo, pance e alettone posteriore) l'alettone anteriore. La vettura ha partecipato alla competizione in Italia nella Formula ATA ed in Ungheria nella Formula Student EAST nel 2016.



Figura 1-5. Mockup Peacock 3

Peacock 3 EVO: Evoluzione della vettura del 2016, la Peacock 3 EVO, presenta migliorie generali sulla vettura e un livello progettuale più avanzato rispetto alla precedente, oltre al nuovo motore Smart che passa da 600cc a 700cc per una modifica del regolamento. La P3 EVO ha partecipato a Formula Student Netherlands 2017, ad Assen.



Figura 1-6. Mockup Peacock 3 EVO

Peacock 4: Penultima vettura del team, che vede stravolto e rivoluzionato il progetto sotto diversi sistemi, si passa infatti da un telaio a traliccio ad un monoscocca in fibra di carbonio ed *honeycomb* in alluminio (struttura a sandwich). Da un motore automobilistico pluricilindrico sovralimentato ad uno di derivazione motociclistica monocilindrico, mantenendo la sovralimentazione. Il pacchetto aerodinamico viene ottimizzato maggiormente grazie agli sviluppi dello stesso in galleria del vento mediante modellino in scala ed il sistema di comunicazione elettronico adottato usa il protocollo CAN-BUS. La vettura ha partecipato alle competizioni di Formula Student Italy 2019 e, per la prima volta, a Formula Student Germany 2019.



Figura 1-7. Mockup Peacock 4

Nel 2023, la squadra si è dilettata nella realizzazione di un veicolo elettrico, denominato Peacock 5 (la quinta generazione del veicolo), portando avanti un progetto biennale che vedrà la partecipazione nella categoria *electric* (Classe 1E) a fine realizzazione. Nel luglio dello stesso anno, il team ha gareggiato nel circuito di Varano de' Melegari (PR) in classe 3, partecipando agli eventi di design, business e cost, classificandosi terza su sei squadre partecipanti.



Figura 1-8. Mockup della Peacock 5

Capitolo 2
INTRODUZIONE ELETTRICA DEL VEICOLO

La Peacock 5 è una vettura moderna che punta al soddisfacimento dei seguenti *drivers*:

- **Innovazione**: le componenti sono state ingegnerizzate appositamente per trovare soluzioni non ancora utilizzate.
- **Facilità di montaggio**: ogni componente deve essere progettato per un facile montaggio e garantire la massima compatibilità con le altre parti.
- **Integrazione efficace**: il design delle componenti deve essere modulare e scalabile.
- **Peso**: ogni componente deve essere scelto in base al suo peso poiché l'obiettivo finale è di avere un veicolo il più leggero possibile.
- **Costo**: le componenti devono garantire la massima qualità in funzione di un costo minore con l'obiettivo di ottenere un veicolo economicamente sostenibile.

2.1 **Architettura del sistema elettrico**

Ci sono svariati sensori presenti sulla macchina che raccolgono dati di differente natura fisica. Le informazioni vengono acquisite dalle schede elettroniche (Feather M4) dislocate in tutta la vettura, le quali analizzano i dati ed eseguono delle azioni se necessario. Ogni scheda (MCU, VCU, PDM) ha le proprie responsabilità. I dati sono resi accessibili a tutte le schede per mezzo di una comunicazione CAN BUS. Ogni scheda che spedisce informazioni sul CAN BUS è identificata univocamente tramite un indirizzo deciso in fase progettuale. Inoltre, la vettura presenta un sistema di telemetria che permette di raccogliere ed immagazzinare i dati a distanza, consentendo la valutazione di essi anche da remoto quindi rimuovendo la necessità di agire sulla macchina solo quando essa è ferma.

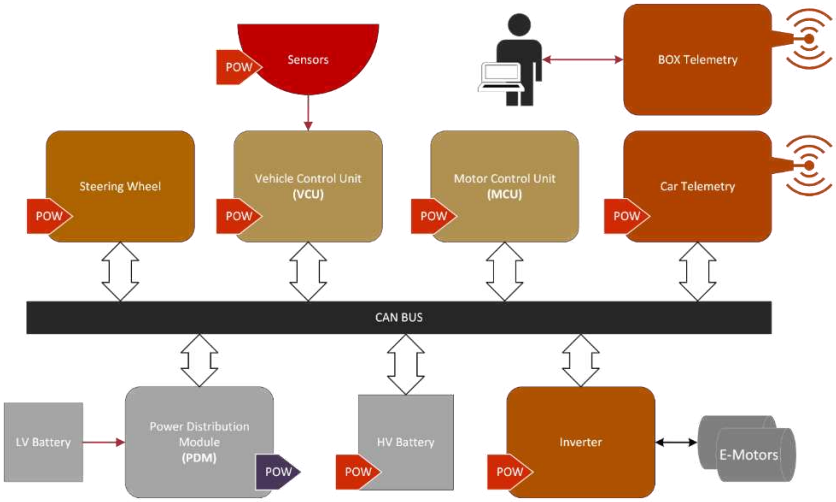


Figura 2-1. Architettura elettrica generale

2.2 **Logica del layout elettrico**

Il layout elettrico è ingegnerizzato in maniera tale da ridurre drasticamente il numero di collegamenti elettrici e ridurre la lunghezza dei cavi. A tal proposito, i sensori e le schede sono posizionati in maniera tale da essere i più vicini possibile per avere una maggiore organizzazione degli spazi e per rendere più facile la manutenzione.

Lo schema elettrico del veicolo è suddiviso in due principali zone che sono la parte anteriore (*front*) e la parte posteriore (*rear*) messe in comunicazione tra loro tramite CAN BUS.

Nella parte di *front* troviamo:

- **VCU anteriore:** è un'unità di controllo del veicolo che svolge una serie di funzioni. Innanzitutto, gestisce la comunicazione con la MCU, o Motor Control Unit, che controlla il motore del veicolo. Inoltre, riceve gli input da vari sensori e trasduttori installati sul veicolo, ad eccezione di quelli che provengono dagli inverter e dal motore. Questi dati includono la temperatura dell'acqua, lo stato dei freni, la temperatura delle gomme e così via. La VCU mostra quindi queste informazioni al pilota tramite un display, in modo che possa tenere d'occhio lo stato attuale della macchina. Infine, nella VCU viene incluso anche il modulo GPS utile per determinare la posizione corrente del veicolo.
- **Sistema di telemetria:** per mezzo di modulo Lo-Ra[1] e modulo Bluetooth, i dati raccolti dai sensori vengono trasmessi in tempo reale al team di ingegneri di supporto che, dopo averli elaborati ed analizzati, prendono decisioni sulle strategie e apportano eventuali modifiche per migliorare le prestazioni del veicolo durante le gare. Per garantire la continuità della raccolta dei dati qualora si verificassero problemi nella comunicazione wireless, la macchina è dotata di un sistema di *SD data logging*, cioè di file di testo salvati in una memoria esterna direttamente collegata alla scheda di acquisizione. Questo sistema è particolarmente utile per analizzare i parametri del veicolo in fase post-gara e per recuperare le informazioni non trasmesse a causa di problemi di comunicazione wireless.

Nella parte di *rear* troviamo:

- **VCU posteriore:** le responsabilità sono simili a quelle della VCU anteriore ma i dati provengono, appunto, da sensori posti sulla parte anteriore del veicolo.
- **PDM:** il Power Distribution Module è la componente responsabile della distribuzione dell'alimentazione elettrica alle diverse periferiche del veicolo. Questo modulo viene programmato per regolare l'azione delle pompe dell'acqua, controllare le ventole di raffreddamento, e gestire la luce del freno in base ai dati che riceve dalle altre schede dislocate sul veicolo. Inoltre, monitora il flusso di corrente e protegge il veicolo da eventuali sovraccarichi o cortocircuiti, oltre al fatto che gestisce l'erogazione delle correnti in modo ottimale, al fine di minimizzare le dimensioni del pacco batteria e, di conseguenza, ridurre il peso complessivo della macchina.

[1] Modulo Lo-Ra: dispositivo di comunicazione wireless basato sulla tecnologia LoRaWan (Long Range Wide Area Network). Questi moduli consentono di trasmettere dati a lungo raggio utilizzando una bassa potenza di trasmissione. Sono costituiti da un trasmettitore e da un ricevitore, che utilizzano una modulazione "chirp" per trasmettere i dati su una banda di frequenza libera: i dati possono attraversare ostacoli senza perdita di segnale.

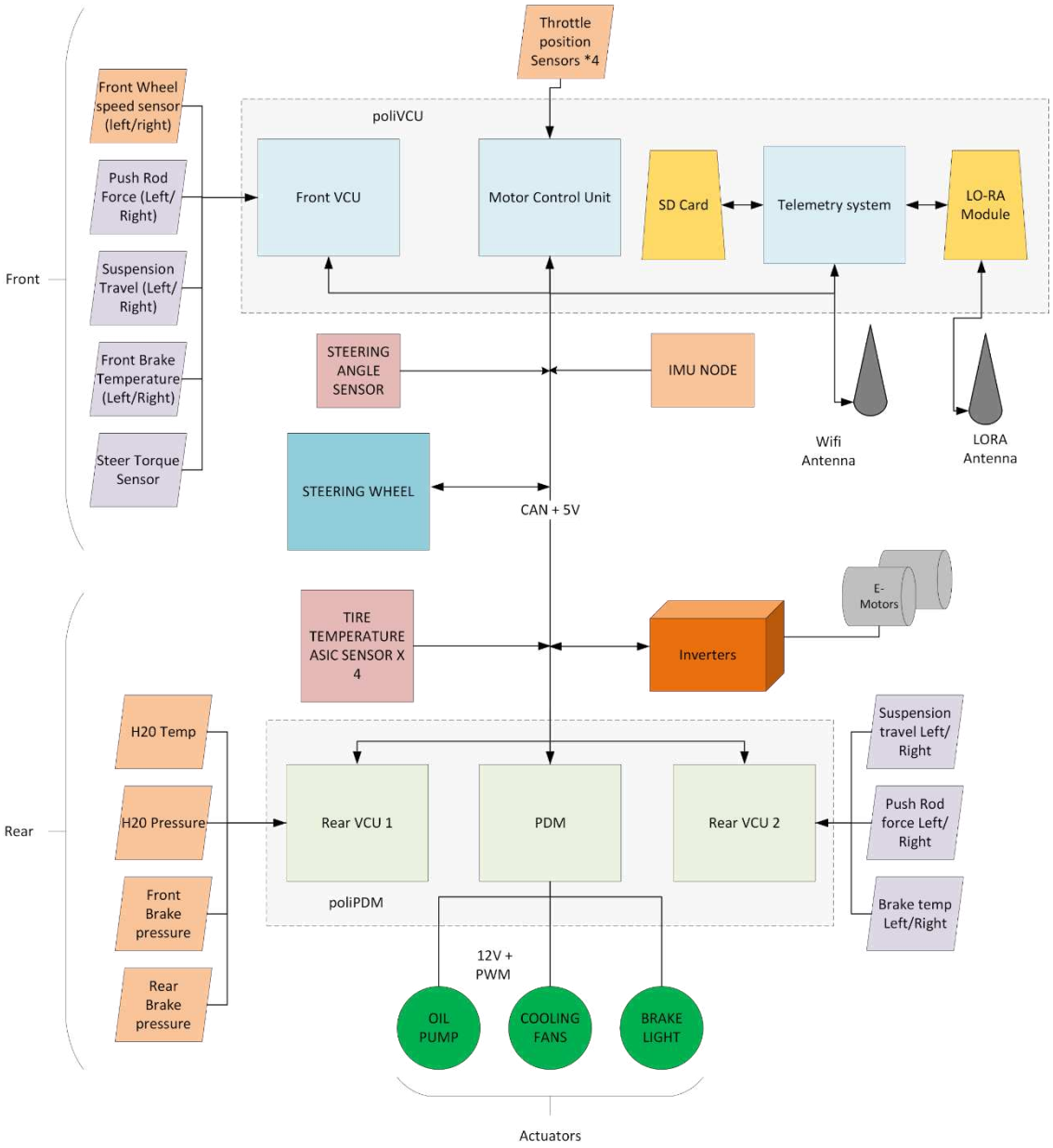


Figura 2-2. Architettura elettrica del veicolo

2.3 **Sensori**

I sensori impiegati nella realizzazione del veicolo consentono di monitorare i parametri vitali della vettura e quindi migliorare le prestazioni, l'efficienza e ridurre il consumo di energia. In Figura 2-3 viene riportata la lista completa dei trasduttori suddivisi per categorie (Temperature, Pressione e Carichi, Tensioni e Voltaggi) e con alcuni parametri caratteristici (Range di valori letti, tipologia di sensore)

Figura 2-3. Lista dei sensori

Temperature					
N°	Label Name	Description	Range	Units	Sensor type
T5.0	T_CoolantIn	Coolant temp before radiator	0 - 80	° C	PT / PTC
T5.1	T_CoolantOut	Coolant temp after radiator	0 - 80	° C	PT / PTC
T6.0	T_PDM1	DC/DC 12/24 Temperature	0-50	° C	NTC
T6.1	T_PDM2	DC/DC 12/5 Temperature	0-50	° C	NTC
T9.0	T_BrakeFR	Brake disc temperature	0 - 650	° C	IR
T9.1	T_BrakeFL	Brake disc temperature	0 - 650	° C	IR
T9.2	T_BrakeRR	Brake disc temperature	0 - 650	° C	IR
T9.3	T_BrakeRL	Brake disc temperature	0 - 650	° C	IR
T10.0	T_TireFR	Tire temperature	0 -90	° C	IR
T10.1	T_TireFL	Tire temperature	0 - 90	° C	IR
T10.2	T_TireRR	Tire temperature	0 - 90	° C	IR
T10.3	T_TireRL	Tire temperature	0 - 90	° C	IR
Pressure & Load					
N°	Label Name	Description	Range	Units	Sensor type
P1	P_BrakeF	Brake pressure, front circuit	0 - 150	Bar	Piezoresistive
P2	P_BrakeR	Brake pressure, rear circuit	0 - 150	Bar	Piezoresistive
P3	P_Coolant	Coolant pressure	0 - 2	Bar	Piezoresistive
Speed & Position					
N°	Label Name	Description	Range	Units	Sensor type
S1	S_WheelFR	Ground speed	0 - 50	m/s	Hall
S2	S_WheelFL	Ground speed	0 - 50	m/s	Hall
PO1	PO_SteeringAngle	Steering angle	± 90	°	Rotary potentiometer
PO2.0	PO_SuspensionFR	Suspension travel	0 - 80	mm	Linear potentiometer
PO2.1	PO_SuspensionFL	Suspension travel	0 - 80	mm	Linear potentiometer
PO2.2	PO_SuspensionRR	Suspension travel	0 - 80	mm	Linear potentiometer
PO2.3	PO_SuspensionRL	Suspension travel	0 - 80	mm	Linear potentiometer
PO3	PO_Throttle	Throttle position	0 - 45	°	Rotary potentiometer
PO4	PO_GPS	GPS	/	/	PCB
A1	A_Accelerometer	Accelerometer	/	/	PCB
Current & Voltage					
N°	Label Name	Description	Range	Units	Sensor type
C0	C_TSAC	TSAC drawn current	0-200	A	Hall Effect

C1	C_LVBattery	Low voltage battery drawn current	0-50	A	PCB
C2	C_12Vout	12 Volt exit current	0-20	A	PCB
C3	C_24V	24V step up drawn current	0-20	A	PCB
C4	C_5V	5V step down drawn current	0-20	A	PCB
C5.1	C_Power_Exit_1	Water pump current drawn at PWM exit	0-20	A	PCB
C5.2	C_Power_Exit_2	Oil pump drawn at PWM exit	0-20	A	PCB
C5.3	C_Power_Exit_3	Fan current drawn at PWM exit	0-20	A	PCB
C5.4	C_Power_Exit_4	Fan current drawn at PWM exit	0-20	A	PCB
V1	V_LVBattery	Low voltage battery Voltage	0-15	V	PCB
V2	V_24VSource	Voltage at 24V step-up exit	0-26	V	PCB
V3	V_5VSource	Voltage at 5V step-down exit	0-7	V	PCB

2.4 **Schede di acquisizione e ambiente di sviluppo**

La scheda di acquisizione utilizzata all'interno del veicolo è la Feather M4 CAN Express, composta da 21 pin GPIO, di cui:

- 12 sono pin digitali,
- 9 sono pin analogici,
- 10 dei pin digitali possono essere utilizzati come uscite PWM,
- 3 dei pin analogici possono essere usati come pin ADC.

La scheda supporta nativamente la comunicazione CAN BUS.

Il software di gestione delle schede è interamente sviluppato utilizzando il linguaggio C/C++ con l'IDE di Visual Studio Code implementando l'utilizzo del framework open source PlatformIO che semplifica lo sviluppo di progetti IoT mettendo a disposizione una varietà di schede e microcontrollori preconfigurati.

Capitolo 3
SOFTWARE PDM

L'obiettivo del software per la gestione del PDM (Power Distribution Module) è di acquisire correnti, tensioni e temperature provenienti dai sensori ad esso collegati e/o dai dati recuperati sul CAN BUS, al fine di ottimizzare gli sprechi di energia e di ridurre le dimensioni del pacco batteria.

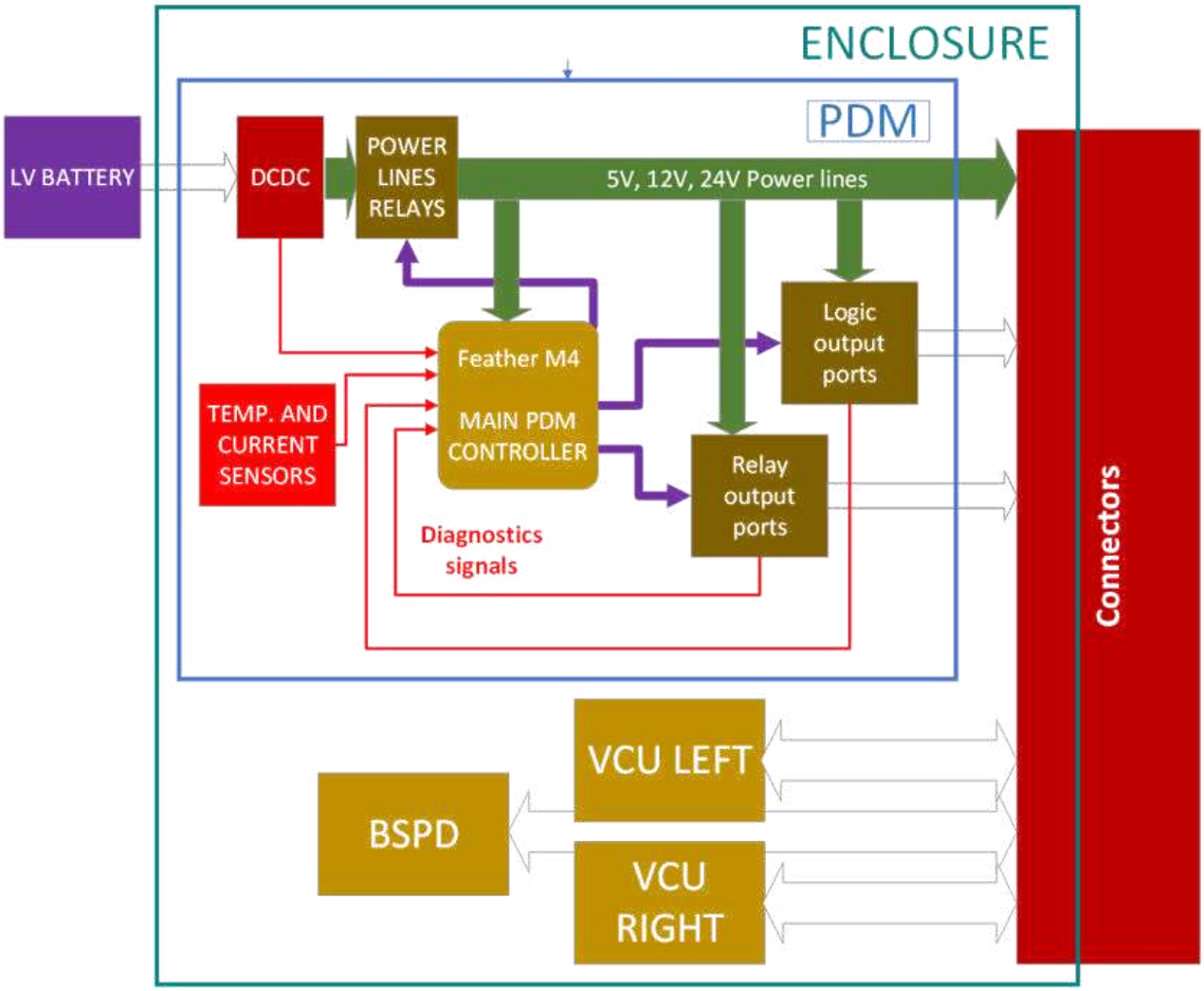


Figura 3-1. Schema logico PDM

Come si può notare dalla Figura 3-1, si compone principalmente di:

- Sensori di temperatura, correnti e tensioni che comunicano direttamente con la scheda.
- Relè sulle linee di potenza.

- Convertitore DC/DC che riduce la tensione da 12V ad una inferiore di 5V.
- Convertitore DC/DC che amplifica la tensione da 12V ad una maggiore di 24V.
- Uscite PWM da 5V e 12V.
- Connettore per l'organizzazione dei cavi e unico input/output.

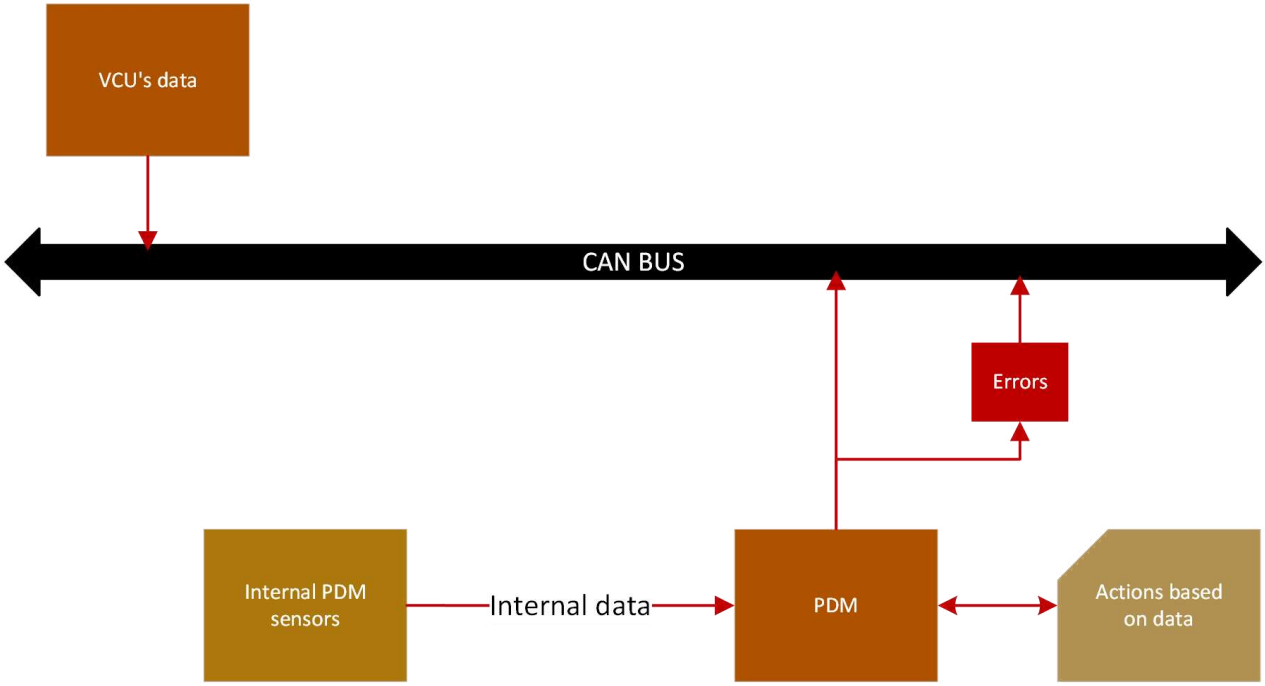


Figura 3-2. Esempio di gestione dei dati del PDM

In Figura 3-2 viene mostrato un esempio di come vengono gestiti i dati dal PDM. Inizialmente la VCU invia sul CAN BUS informazioni quali: temperature, percentuali di duty cycle per il PWM, segnali per gestire i freni ed errori. Successivamente il PDM andrà a recuperare dal CAN BUS le informazioni. Contestualmente anche i sensori che sono direttamente collegati alla scheda di acquisizione del PDM invieranno dati. Il software, in base alle informazioni ricevute, eseguirà delle azioni e spedirà i risultati sul CAN BUS, eventualmente con i messaggi di errore se ce ne fossero.

3.1 **Acquisizione dati**

Di seguito vengono mostrati i lavori svolti per la gestione software del PDM riguardante l'acquisizione dei dati e l'elaborazione di essi.

3.1.1 *Acquisizione delle temperature*



Figura 3-3. Sensori di temperatura

L'obiettivo del software è quello di acquisire le temperature dei due convertitori DC/DC montati sulla scheda PDM. Viene letta la tensione sul partitore formato dal termistore NTC[2] ed un resistore da 10kΩ. Il risultato ottenuto dovrà essere convertito nella temperatura in gradi celsius corrispondente.

Per ottenere la relazione tra la temperatura e la resistenza data dal termistore NTC, viene sfruttata l'equazione di Steinhart-Hart:

$$\frac{1}{T} = \ A + B * \ln\left(\frac{R}{R0}\right) + C * \ln\left(\frac{R}{R0}\right)^3$$

Data la sua complessità algebrica, viene fatto uso della B-parameter Equation che deriva direttamente dalla precedente tramite alcune manipolazioni algebriche:

$$\frac{1}{T} = \frac{1}{T0} + \left(\frac{1}{B}\right) * \ln\left(\frac{R}{R0}\right)$$

I parametri T0, B, R e R0 sono rispettivamente la temperatura nominale (25° = 298.15K), il coefficiente β (3950.0) specifico di questo modello NTC, la resistenza calcolata sul partitore e la resistenza calcolata tenendo conto del valore nominale della resistenza del termistore.

[2] Un sensore NTC, o termistore negativo, è un dispositivo utilizzato per misurare la temperatura. È composto da un materiale semiconduttore che ha un coefficiente di temperatura negativo ossia la sua resistenza elettrica diminuisce all'aumentare della temperatura.

3.1.2 *Acquisizione delle correnti*

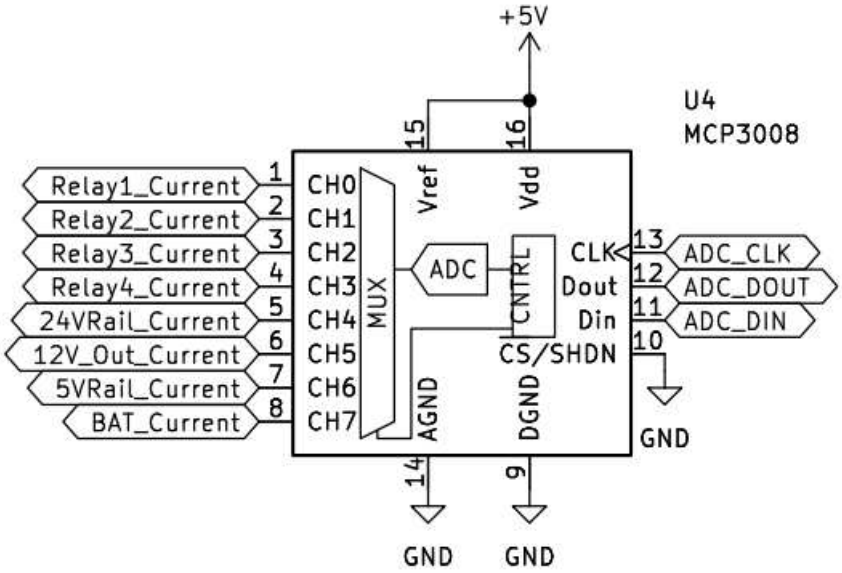


Figura 3-4. Correnti sull'ADC esterno

Il software deve fornire il valore delle correnti che attraversano la scheda PDM. Viene impiegato un MPC3008[3] per leggere le tensioni di uscita provenienti dai sensori di corrente. Questo ADC esterno è composto da otto canali ognuno dei quali rappresenta un dato specifico.

- Canali 1-4: correnti che provengono dai relè.
- Canali 5-7: correnti che provengono dalle rail di alimentazione 5V, 12V, 24V.
- Canale 8: corrente che proviene dal pacco batteria.

Dopo aver letto la tensione (V) di ogni canale, conoscendo il parametro di sensibilità di ogni sensore (R), risulta facile calcolare la corrente tramite la formula $I = \frac{V}{R}$.

Si può risparmiare l'utilizzo di un canale trovando la corrente di una delle rail tramite la Legge di Kirchoff. Ad esempio, la corrente sulla rail 5V si ottiene sottraendo le correnti calcolate sulle rails 24V e 12V.

[3] Un ADC è un convertitore di segnale da analogico a digitale e funziona misurando l'ampiezza di un segnale analogico ad intervalli temporali regolari (campioni). La frequenza di campionamento (ossia il numero di campioni al secondo) determina la precisione della conversione. L'MCP3008 ha un'ampiezza di ingresso di 0-5 volt e una risoluzione di 10 bit, ciò significa che può convertire un segnale analogico in un valore digitale con una precisione di 1/1024.

3.1.3 *Acquisizione delle tensioni*

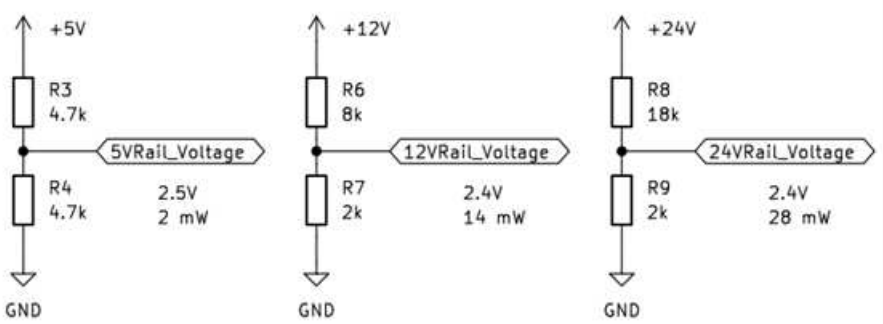


Figura 3-5. Sensori di tensione

Il software mira ad acquisire i valori di tensione delle varie rails di alimentazione. Questi valori sono letti sui pin:

- A2 per la rail 5V.
- A3 per la rail 12V.
- A4 per la rail 24V.

Per mezzo dei partitori di tensione[4] si può notare come le tensioni vengono ridotte rispettivamente di $\frac{1}{2}, \frac{1}{5}, \frac{1}{10}$ per essere lette dall'ADC interna alla Feather M4 che ha risoluzione 12 bit e lettura massima di 3.3V. Valutando la risoluzione dell'ADC allora ogni unità equivale a 0.8mV[5], perciò per ottenere la tensione misurata è necessario moltiplicare il valore letto dall'ADC per 0.0008V e quindi moltiplicare per l'inverso del rapporto di divisione del partitore.

[4] Un partitore di tensione è un circuito costituito da più componenti passivi sui quali si fa ripartire la tensione applicata. Prelevando la tensione dai capi di uno solo dei componenti, si ottiene ovviamente una tensione più bassa. I calcoli della riduzione delle tensioni sono stati fatti tenendo conto della relazione $\alpha = \frac{R2}{R1+R2}$ della formula $V_{out} = \alpha * V_{in}$

[5] Questo calcolo deriva dal fatto che, se si ha una risoluzione di 12 bit per misurare una tensione di 3.3V, allora si può calcolare la corrispondenza tra ciascun bit e la tensione sfruttando la formula $\frac{Tensione\ massima}{Numero\ di\ bit} = volts\ per\ bit$. Quindi, ogni bit rappresenta circa 0.0008057V (o 0.8057mV) in questo contesto specifico.

3.1.4 *Controllo dei relè*

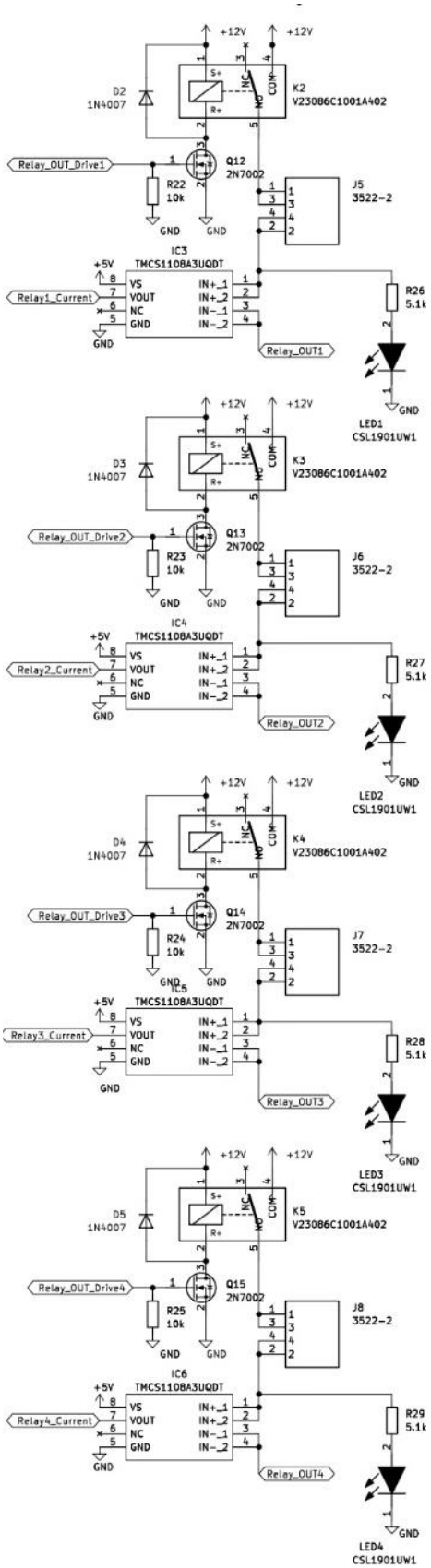


Figura 3-6. Schema dei relè

Il software deve gestire l'apertura/chiusura dei relé in seguito a dei messaggi acquisiti direttamente dal CAN BUS inviati appositamente dalla VCU. Questi messaggi sono:

- ON/OFF delle pompe dell'acqua.
- ON/OFF delle ventole di raffreddamento
- ON/OFF del convertitore 24V (inverter)
- ON/OFF dell'uscita che alimenta l'elettronica a 12V

Quindi, gli input del programma sono il numero del pin del relé che si vuole controllare e lo stato che si vuole impostare. Ovviamente, OFF indica che deve essere tagliata l'alimentazione sul relé mentre ON il contrario. I pin coinvolti sono:

- D9 (Relay_OUT_Drive3)
- D10 (Relay_OUT_Drive2)
- D11 (Relay_OUT_Drive1)
- A5 (Relay_OUT_Drive4)
- D12 (Relay12_Connector)
- D13 (Relay24V)

3.1.5 *Generazione segnali PWM e segnali luce dei freni*

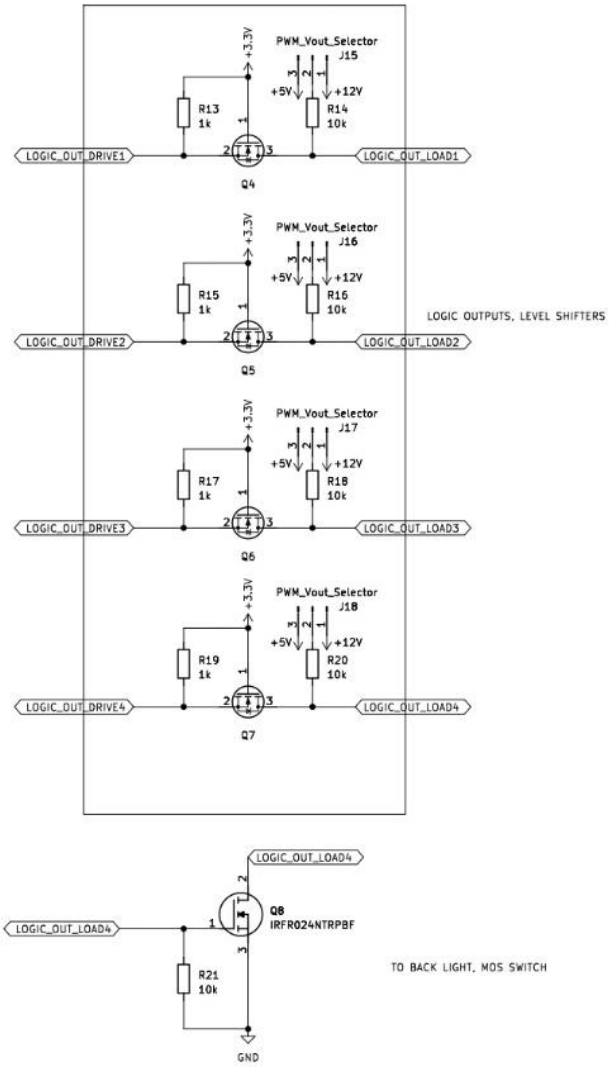


Figura 3-7. Sensori PWM e luce freni

Il software ha l'obiettivo di generare segnali PWM su dei pin prestabiliti, i quali sono:

- D0 (PWM_OUT_DRIVE_1)
- D1 (PWM_OUT_DRIVE_2)
- D2 (PWM_OUT_DRIVE_3)
- D4 (PWM_OUT_DRIVE_4)
- D6 (PWM_OUT_DRIVE_BRAKELIGHT)

Le informazioni, prelevate dal CAN BUS e provenienti principalmente dalla VCU, sono le percentuali di duty cycle richieste per ognuno di questi pin specifici. Quindi, l'output è un segnale PWM ad una frequenza prefissata e con il duty cycle richiesto. In caso di temperature troppo elevate, per prevenire la rottura di componenti o l'errato funzionamento del veicolo, viene trasmesso in input un livello di duty cycle pari al 100% (*full signal*) così da avere il sistema di raffreddamento al massimo della sua potenza.

Infine, il software gestisce l'accensione/spegnimento della luce del freno in base alla pressione sul pedale da parte del pilota. Viene stabilita la seguente convenzione:

- Pressione sul freno indica che la luce deve accendersi. Viene spedito su CAN BUS un valore di duty cycle pari al 100%.
- Nessuna pressione sul freno indica che la luce deve spegnersi. Viene spedito su CAN BUS un valore di duty cycle pari allo 0%.

3.1.6 *Librerie C++*

Temperature.h

```
#ifndef PDM_TEMPERATURE_ACQUISITION
#define PDM_TEMPERATURE_ACQUISITION

#include <math.h>
#include <NTC_871B57702M0103J000.h>
#include <Feather_M4_CAN.h>

class Temperatures
{
private:
    int thermistorPin;
    const float SERIES_RESISTOR = 10000.0; // Valore della resistenza in serie al
termistore (ohm)
    float getResistance();

public:
    Temperatures(int pin = THERMISTOR_PIN); //THERMISTOR_PIN = A0
    float getTemperatures();
};

#endif
```

Temperature.cpp

```
#include <Temperature.h>

Temperatures::Temperatures(int pin) : thermistorPin(pin) {}

// Calcolo della tensione di ingresso
// Calcolo della resistenza Vout = Vin*R2/ (R1+R2) ==> R1 = R2 *(1-Vout/Vin)
float Temperatures::getResistance()
{
    uint16_t adcValue = analogRead(thermistorPin);
    float voltage = adcValue * M4_V_REF / M4_RESOLUTION;
    return SERIES_RESISTOR * (M4_V_REF / voltage - 1.0);
}

float Temperatures::getTemperatures()
{
    float resistance = getResistance(), steinhart;
    steinhart = resistance / THERMISTOR_NOMINAL;       // (R/Ro)
    steinhart = log(steinhart);                        // ln(R/Ro)
    steinhart /= BCOEFFICIENT;                         // 1/B * ln(R/Ro)
    steinhart += 1.0 / (TEMPERATURE_NOMINAL + 298.15); // + (1/To)
    steinhart = 1.0 / steinhart;                       // Inverti
    return steinhart - 298.15;                         // Converti in °C
}
```

Currents.h

```
#ifndef PDM_CURRENT_ACQUISITION
#define PDM_CURRENT_ACQUISITION

#include <MCP3008.h>

class Currents
{

private:
    const float BATTERY_SENSITIVITY = 0.04; // 40mV/A
    const float OTHERS_SENSITIVITIES = 0.2; // 200mV/A
    const float V_OFFSET = 0.5;             // 500mV
    typedef struct
    {
        int relayPin1;
        int relayPin2;
        int relayPin3;
        int relayPin4;
        int rail5VPin;
        int rail24VPin;
        int batteryPin;
    } PinMap;

    PinMap pinMap = {0, 1, 2, 3, 5, 4, 6};

public:
    float voltages[CHANNELS - 1];
    float currents[CHANNELS];
    Adafruit_MCP3008 mcp;
    void readVoltages();
    void toCurrents();
};

#endif
```

Currents.cpp

```cpp
#include <Current.h>


// utilizziamo solo 7 canali di ingresso, quindi CHANNELS-1
// Legge le tensioni da tutti i canali e li memorizza in un array di tensioni
void Currents::readVoltages()
{
    for (int i = 0; i < CHANNELS - 1; i++)
    {
        voltages[i] = mcp.readADC(i) * (MCP_V_REF / MCP_RESOLUTION);
    }
}

/*
    Ciclo for per tutti i primi 5 canali.
    Converto le tensioni in corrente conoscendo la sensibilità e la tensione di offset
dei sensori .
    Infine, calcolo la corrente della rail 12V per differenza e la corrente della
batteria.

    Indicizzazione dell'array delle correnti:
    [0] -> relay 1
    [1] -> relay 2
    [2] -> relay 3
    [3] -> relay 4
    [4] -> rail 24V
    [5] -> rail 5V
    [6] -> rail 12V
    [7] -> batteria
  */
void Currents::toCurrents()
{
    for (int i = 0; i < CHANNELS - 2; i++)
    {
        // corrente dei primi 5 canali
        currents[i] = (voltages[i] - V_OFFSET) / OTHERS_SENSITIVITIES;
    }

    // corrente della rail 12V
    currents[CHANNELS - 2] = currents[pinMap.rail24VPin] - currents[pinMap.rail5VPin];
    // corrente della batteria
    currents[CHANNELS - 1] = voltages[pinMap.batteryPin] / BATTERY_SENSITIVITY;
}
```

Voltages.h

```
#ifndef PDM_VOLTAGE_ACQUISITION
#define PDM_VOLTAGE_ACQUISITION

#include <Arduino.h>
#include <Feather_M4_CAN.h>

class Voltages
{
private:
    const int TOT_CURRENTS = 3;

public:
    const float RATIO_5V = 2.0;
    const float RATIO_12V = 5.0;
    const float RATIO_24V = 10.0;
    const int PIN_5V = A2;
    const int PIN_12V = A3;
    const int PIN_24V = A4;
    float getVoltage(int pin, float voltageDividerRatio);
};

#endif
```

Voltages.cpp

```
#include <Voltage.h>

float Voltages::getVoltage(int pin, float voltageDividerRatio)
{
    return (analogRead(pin) * (M4_V_REF / M4_RESOLUTION)) * voltageDividerRatio;
}
```

PWMHandler.h

```
#ifndef PDM_PWM_HANDLER
#define PDM_PWM_HANDLER

#include <Arduino.h>
#include <SAMD_PWM.h>

class PWMHandler
{
public:
    int NUM_OF_PINS = 5;
    void sendPWMSignal(float *dutyCycles, float *frequencies );

private:
    uint32_t PWM_Pins[5] = {0, 1, 4, 5, 6};
    SAMD_PWM* PWM_Instance[5];
};

#endif
```

PWMHandler.cpp

```
#include <PWMHandler.h>

void PWMHandler::sendPWMSignal(float *dutyCycles, float *frequencies)
{
    for (uint8_t index = 0; index < NUM_OF_PINS; index++)
  {
    PWM_Instance[index] = new SAMD_PWM(PWM_Pins[index], frequencies[index],
dutyCycles[index]);

    if (PWM_Instance[index])
    {
      PWM_Instance[index]->setPWM();
    }
  }
}
```

RelayHandler.h

```
#ifndef PDM_RELAY_HANDLER
#define PDM_RELAY_HANDLER

#include <Arduino.h>

class RelayHandler
{
public:
    const int PIN5 = A5;          //Relay_OUT_Drive4
    const int PIN9 = 9;           //Relay_OUT_Drive3
    const int PIN10 = 10;         //Relay_OUT_Drive2
    const int PIN11 = 11;         //Relay_OUT_Drive1
    const int PIN12 = 12;         //Relay12_Connector
    const int PIN13 = 13;         //Relay24V
    void setRelayState(bool state, int pin);
};

#endif
```

RelayHandler.cpp

```
#include <RelayHandler.h>

void RelayHandler::setRelayState(bool state, int pin)
{
    if (state == 1)
    {
        digitalWrite(pin, HIGH);
    }
    else
    {
        digitalWrite(pin, LOW);
    }
}
```

3.2 **Comunicazione CANBUS**

All'inizio degli anni '80, i produttori di veicoli iniziarono a incorporare per la prima volta nelle automobili e altre vetture un numero sempre crescente di dispositivi elettronici, come il controllo del cambio e dell'illuminazione, chiusura centralizzata e ABS e così via. Affinché questi dispositivi funzionino all'unisono, cronometrino correttamente le attività e condividano i dati, dovrebbero essere cablati insieme. In base agli standard di cablaggio già esistenti, le componenti elettroniche comunicavano tra loro utilizzando linee di segnale analogiche punto-punto. Era un'architettura che richiedeva molto tempo nel realizzarla e utilizzava una quantità eccessiva di cavi.

Grazie a Robert Bosch viene introdotto un nuovo tipo di standard di comunicazione, il protocollo CAN, il quale elimina la necessità di un cablaggio eccessivo consentendo ai dispositivi di comunicare tra loro lungo un singolo cavo (architettura half-duplex) che collega ciascun nodo della rete con gli altri. Con tale architettura, quindi, ogni componente elettronico è in grado di ricevere ed inviare dati, ma non può fare entrambe le operazioni contemporaneamente in quanto il canale di comunicazione è unico.

3.2.1 *Funzionamento del CAN*

Per stabilire una connessione fisica tra due dispositivi tramite il CAN BUS bastano due cavi: CAN High (CANH) e CAN Low (CANL). Il protocollo utilizza una tecnica chiamata *differenziale* per l'inoltro dei dati dove, durante la trasmissione, il segnale sul CANH sarà complementare al segnale sul CANL. Questo approccio aiuta a ridurre l'interferenza elettromagnetica (EMI) e rende il sistema più resistente ai disturbi esterni.

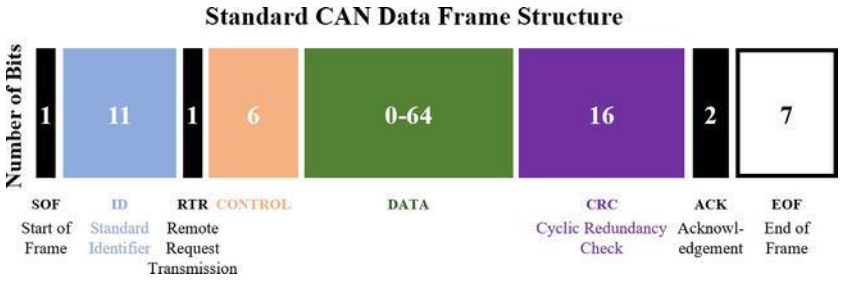


Figura 3-8. Struttura di un messaggio CAN BUS

Come mostrato in Figura 3-8, la struttura di un messaggio che viaggia sul CAN BUS ha diverse componenti:

- **SOF:** è uno "zero dominante" che occupa un bit ed è posto all'inizio di un messaggio CAN. Segnala che un nodo sta per inviare un messaggio quindi gli altri dispositivi si devono mettere in ascolto.
- **ID:** contiene informazioni sulla priorità del messaggio (i valori più bassi indicano una priorità più alta) e l'indirizzo di origine. La lunghezza del bit dell'identificatore varia in base alla versione che si utilizza: per esempio con il CAN 2.0 si hanno 11 bit a disposizione, mentre per le versioni successive si arriva a disporre di 29 bit.
- **RTR:** è un frame del messaggio che occupa solo un bit e indica se un nodo sta inviando o richiedendo dati a uno specifico dispositivo connesso al bus.

- **Control:** questo frame occupa 6 bit totali, dove 4 di questi denotano la dimensione del messaggio che verrà trasmesso (DLC).
- **Data:** contiene l'informazione che dovrà essere trasmessa e letta dagli altri dispositivi.
- **CRC:** ha la responsabilità di controllare l'integrità del messaggio.
- **ACK:** è un'informazione che indica che il CRC non ha trovato problemi.
- **EOF:** segnala la fine della trasmissione.

Uno dei punti di forza del CAN BUS è la gestione degli errori. Quando si verifica un errore di trasmissione, il nodo che l'ha rilevato ha la responsabilità di inviare un *flag di errore* a tutti gli altri dispositivi connessi i quali ritrasmettono 8 bit recessivi, chiamati *delimitatori di errore*, che servono a pulire il BUS prima di eseguire le azioni per contrastare l'errore. La risposta più comune a un *flag di errore* è di scartare il messaggio errato e continuare a trasmettere e ricevere i dati sul bus. Ciò consente la *tolleranza agli errori* quindi la capacità del sistema di funzionare attorno a uno stato di errore.

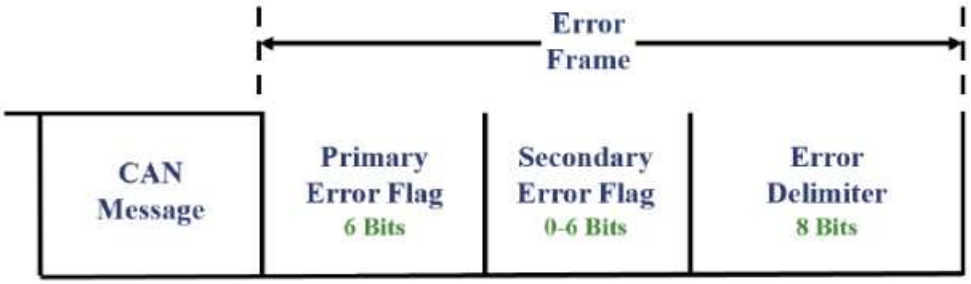


Figura 3-9. Struttura di un messaggio di gestione degli errori

3.2.2 *Progettazione del software*

I dati letti dai sensori e dalle varie altre componenti devono essere disponibili a tutti i nodi connessi. Il prerequisito fondamentale concordato in fase di progettazione è che l'utente deve manipolare il tipo di dato originale (float, double, int) affinché il dato da trasmettere sia un intero (int).

A questo punto:

- Le informazioni vengono trasformate dal tipo intero a valori esadecimali e vengono spediti sul CAN;
- Ogni nodo è sempre in ascolto di possibili messaggi in arrivo. Quando ne viene intercettato uno, si attiva un meccanismo inverso che trasforma i dati esadecimali in tipo intero.

Il payload di un messaggio CANBUS può contenere al massimo 8 byte (0-7) ragion per cui è stato classificato, per ogni singolo indirizzo CAN, quanti byte occupa (Figura 3-10). All'interno di ogni messaggio, lo spazio che può occupare una singola informazione è al massimo 4 byte (ossia $2^{16} = 65536$ informazioni).

		Bytes messaggio							
ID	DLC	0	1	2	3	4	5	6	7
0x001	1								
0x002	1								
0x003	1								
0x010	1	AMK_ERR							
0x011	1	DRS_CMD							
0x012	1	BRAKE_CMD							
0x013	2	WATER_PUMP	WATER_PUMP						
0x014	2	COOL_FAN	COOL_FAN						
0x015	2	OIL_PUMP	OIL_PUMP						
0x016	1	ReadyToDrive							
0x020	1	AMK_DRVR_ENB							
0x021	8	T_BrakeFR	T_BrakeFR	T_BrakeFL	T_BrakeFL	T_TireFR	T_TireFR	T_TireFL	T_TireFL
0x022	2	T_VCU	T_VCU						
0x023	8	T_CoolantIn	T_CoolantIn	T_CoolantOut	T_CoolantOut	T_VCU	T_VCU	T_Air	T_Air
0x024	8	T_BrakeRR	T_BrakeRR	T_BrakeRL	T_BrakeRL	T_TireRR	T_TireRR	T_TireRL	T_TireRL
0x025	4	T_PDM	T_PDM	T_PDM2	T_PDM2				
0x026	2	T_MCU	T_MCU						
0x027	2	T_Water	T_Water						
0x030	8	S_WheelFR	S_WheelFR	S_WheelFL	S_WheelFL	PO_SteeringAngle	PO_SteeringAngle		
0x031	4	PO_SuspensionFR	PO_SuspensionFR	PO_SuspensionFL	PO_SuspensionFL				
0x032	8	PO_GPS	PO_GPS	PO_GPS	PO_GPS	PO_GPS	PO_GPS	PO_GPS	PO_GPS
0x033	6	A_Acc_X	A_Acc_X	A_Acc_Y	A_Acc_Y	A_Acc_Z	A_Acc_Z		
0x034	6	A_Gyr_X	A_Gyr_X	A_Gyr_Y	A_Gyr_Y	A_Gyr_Z	A_Gyr_Z		
0x035	4	PO_Throttle1	PO_Throttle1	PO_Throttle2	PO_Throttle3				
0x040	1	AMK_DC_ON							
0x041	6	L_PushFR	L_PushFR	L_PushFL	L_PushFL	L_SteerTorque	L_SteerTorque		
0x042	8	P_BrakeF	P_BrakeF	P_BrakeR	P_BrakeR	P_CoolantH20	P_CoolantH20	P_CoolantOil	P_CoolantOil
0x043	4	L_PushRR	L_PushRR	L_PushRL	L_PushRL				
0x044	4	PO_SuspensionRR	PO_SuspensionRR	PO_SuspensionRL	PO_SuspensionRL				
0x045	8								
0x050	8	C_LVBattery	C_LVBattery	C_24V	C_24V	C_12Vout	C_12Vout	C_5V	C_5V
0x051	6	V_LVBattery	V_LVBattery	V_24VSource	V_24VSource	V_5VSource	V_5VSource		
0x052	8	C_Power1	C_Power1	C_Power2	C_Power2	C_Power3	C_Power3	C_Power4	C_Power4
0x080	1	AMK_INV_ON							
0x183	?	AMK_S1_BA							
0x184	8	AMK_CTRL	AMK_CTRL	AMK_TV	AMK_TV	AMK_TLP	AMK_TLP	AMK_TLN	AMK_TLN
0x185	8	AMK_CTRL	AMK_CTRL	AMK_TV	AMK_TV	AMK_TLP	AMK_TLP	AMK_TLN	AMK_TLN
0x188	8	AMK_CTRL	AMK_CTRL	AMK_TV	AMK_TV	AMK_TLP	AMK_TLP	AMK_TLN	AMK_TLN
0x189	8	AMK_CTRL	AMK_CTRL	AMK_TV	AMK_TV	AMK_TLP	AMK_TLP	AMK_TLN	AMK_TLN
0x282	?	AV1_BA							
0x283	8	AMK_STATUS	AMK_STATUS	AMK_AV	AMK_AV	AMK_TC	AMK_TC	AMK_MC	AMK_MC
0x284	8	AMK_STATUS	AMK_STATUS	AMK_AV	AMK_AV	AMK_TC	AMK_TC	AMK_MC	AMK_MC
0x285	8	AMK_TM	AMK_TM	AMK_TI	AMK_TI	AMK_EI	AMK_EI	AMK_TIGBT	AMK_TIGBT
0x286	8	AMK_TM	AMK_TM	AMK_TI	AMK_TI	AMK_EI	AMK_EI	AMK_TIGBT	AMK_TIGBT
0x287	8	AMK_STATUS	AMK_STATUS	AMK_AV	AMK_AV	AMK_TC	AMK_TC	AMK_MC	AMK_MC
0x288	8	AMK_STATUS	AMK_STATUS	AMK_AV	AMK_AV	AMK_TC	AMK_TC	AMK_MC	AMK_MC
0x289	8	AMK_TM	AMK_TM	AMK_TI	AMK_TI	AMK_EI	AMK_EI	AMK_TIGBT	AMK_TIGBT
0x290	8	AMK_TM	AMK_TM	AMK_TI	AMK_TI	AMK_EI	AMK_EI	AMK_TIGBT	AMK_TIGBT
0x300	8								
0x301	8								
0x302	8								
0x303	8								
0x304	8								
0x400	6	COUNT	COUNT	DATA1	DATA1	DATA2	DATA2		
0x401	6	COUNT	COUNT	DATA1	DATA1	DATA2	DATA2		
0x402	6	COUNT	COUNT	DATA1	DATA1	DATA2	DATA2		
0x403	6	COUNT	COUNT	DATA1	DATA1	DATA2	DATA2		
0x404	6	COUNT	COUNT	DATA1	DATA1	DATA2	DATA2		

Figura 3-10. Suddivisione dei bytes per indirizzo

A tal proposito, il software è strutturato in maniera tale da poter gestire i seguenti casi:

- Le informazioni da trasmettere occupano 1 byte allora utilizzo il metodo *sendMoreOneByte*;
- Le informazioni da trasmettere occupano 2 byte allora utilizzo il metodo *sendOneByte.*

Per esempio, si voglia inviare due valori interi sul CAN, si otterrebbero i seguenti risultati:

- Indirizzo 0x21. 4 dati che occupano 2 byte:

Nodo trasmittente

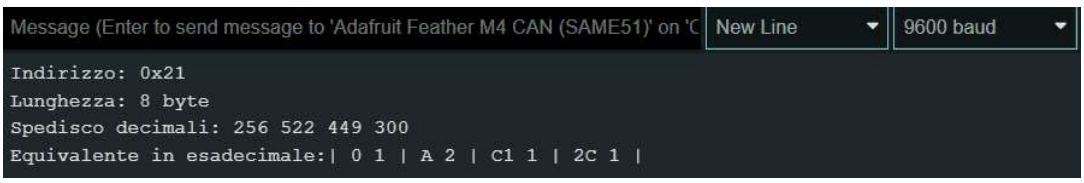


Figura 3-11. Nodo trasmittente, 2 byte per dato

Nodo ricevente

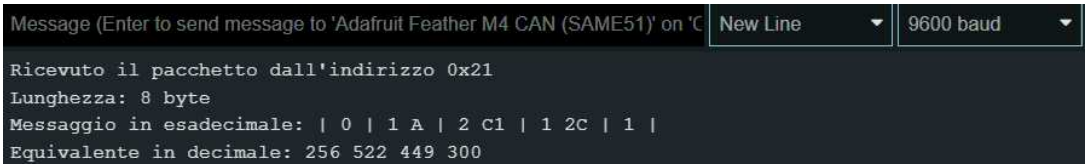


Figura 3-12. Nodo ricevente, 2 byte per dato

- Indirizzo 0x10. 1 dato che occupa 1 byte:

Nodo trasmittente

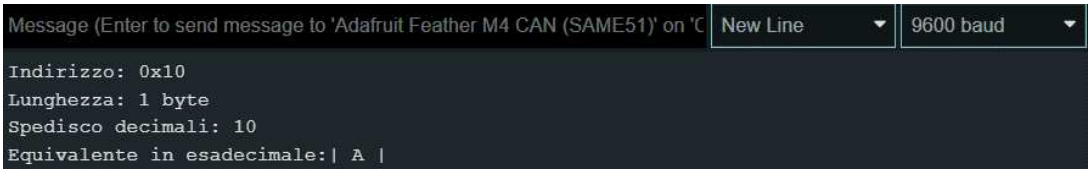


Figura 3-13. Nodo trasmittente, 1 byte per dato

Nodo ricevente

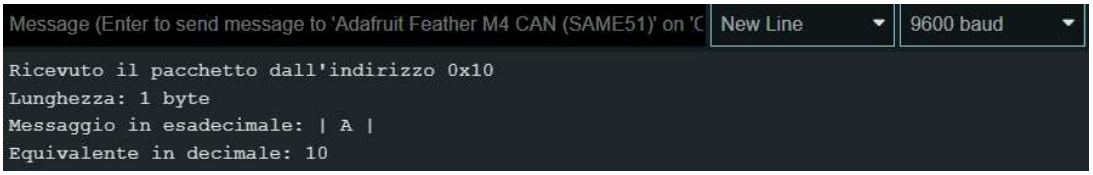


Figura 3-14. Nodo ricevente, 1 byte per dato

3.2.3 *Librerie C++*

Canbus.h

```
#ifndef CANBUS_h
#define CANBUS_h

#include <Arduino.h>

#define PIN_CAN_STANDBY (40)
#define PIN_CAN_BOOSTEN (41)

class CANBUS {
    public:
        void sendMessage(int address, const int* data, int dataLength, bool moreOneByte =
true);
        void setupCAN();
    private:
        void intToBytes(const int* intArray, int arrayLength, uint8_t* byteArray, bool
moreOneByte);
        bool checkArrayLenght(int dataLength);
        bool checkDataBytes(const int* data, int dataLength);
        void sendMoreOneByte(int address, const int* data, int dataLength);
        void sendOneByte(int address, const int* data, int dataLength);

};

void onReceive(int packetSize);
void bytesToInt(const uint8_t* byteArray, int arrayLength, int* intArray, bool
moreOneByte);

#endif
```

Canbus.cpp

```
#include "canbus.h"
#include <CANSAME5x.h>

CANSAME5x CAN;

void CANBUS::setupCAN()
{
    pinMode(PIN_CAN_STANDBY, OUTPUT);
    digitalWrite(PIN_CAN_STANDBY, false); // modalità standby off
    pinMode(PIN_CAN_BOOSTEN, OUTPUT);
    digitalWrite(PIN_CAN_BOOSTEN, true); //modalità boost ON

    if (!CAN.begin(500E3))
    {
        Serial.println("Inizializzazione CAN fallita!");
        while (1)
            delay(10);
    }

    CAN.onReceive(onReceive);
}

// Funzione che converte un array di interi in un array di byte
void CANBUS::intToBytes(const int *intArray, int arrayLength, uint8_t *byteArray, bool
moreOneByte)
{
    if (moreOneByte)
    {
        for (int i = 0; i < arrayLength; i++)
        {
            byteArray[i * 2] = intArray[i] & 0xFF;
            byteArray[i * 2 + 1] = (intArray[i] >> 8) & 0xFF;
        }
    }
    else
    {
        for (int i = 0; i < arrayLength; i++)
        {
            byteArray[i] = intArray[i] & 0xFF;
        }
    }
}

void bytesToInt(const byte *byteArray, int arrayLength, int *intArray, bool moreOneByte)
{
    if (moreOneByte)
    {
        for (int i = 0; i < arrayLength; i++)
        {
            intArray[i] = (byteArray[i * 2 + 1] << 8) | byteArray[i * 2];
        }
    }
    else
    {
        for (int i = 0; i < arrayLength; i++)
        {
            intArray[i] = byteArray[i];
        }
    }
}

bool CANBUS::checkArrayLenght(int dataLength)
```

```
{
    if (dataLength > 4)
    {
        Serial.println("Errore: array di interi con più di 4 elementi.");
        return false;
    }
    return true;
}

bool CANBUS::checkDataBytes(const int *data, int dataLength)
{
    for (int i = 0; i < dataLength; i++)
    {
        // if (data[i] > 65535)
        if (data[i] > 0xFFFF)
        {
            Serial.println("Errore: intero troppo grande. Massimo consentito 2 byte.");
            return false;
        }
    }
    return true;
}

void CANBUS::sendMessage(int address, const int *data, int dataLength, bool moreOneByte)
{
    if (!checkArrayLenght(dataLength) || !checkDataBytes(data, dataLength))
    {
        return;
    }

    if (moreOneByte)
    {
        sendMoreOneByte(address, data, dataLength);
    }
    else
    {
        sendOneByte(address, data, dataLength);
    }
}

// Funzione che invia un messaggio con elementi che occupano più byte
void CANBUS::sendMoreOneByte(int address, const int *data, int dataLength)
{
    uint8_t byteArray[dataLength * 2];

    intToBytes(data, dataLength, byteArray, true);

    CAN.beginPacket(address);
    for (int i = 0; i < dataLength * 2; ++i)
    {
        CAN.write(byteArray[i]);
    }
    CAN.endPacket();
}

// Funzione che invia un messaggio con elementi che occupano un solo byte
void CANBUS::sendOneByte(int address, const int *data, int dataLength)
{
    uint8_t byteArray[dataLength];

    intToBytes(data, dataLength, byteArray, false);

    CAN.beginPacket(address);
    for (int i = 0; i < dataLength; ++i)
```

```
    {
        CAN.write(byteArray[i]);
    }
    CAN.endPacket();
}

// Funzione che viene chiamata quando viene ricevuto un messaggio
void onReceive(int packetSize)
{
    uint8_t receivedBytes[packetSize];
    bool hasMoreThanOneByte = packetSize % 2 == 0;

    // determino la dimensione dell'array di interi
    // se la dimensione del pacchetto è pari, allora l'array di interi avrà metà elementi
(ogni elemento è composto da 2 byte)
    // se la dimensione del pacchetto è dispari, allora l'array di interi avrà tanti
elementi quanti i byte del pacchetto (ogni elemento è composto da 1 byte)
    int intArraySize;

    if (packetSize % 2 == 0)
    {
        intArraySize = packetSize / 2;
    }
    else
    {
        intArraySize = packetSize;
    }

    int intArray[intArraySize];
    bytesToInt(receivedBytes, intArraySize, intArray, hasMoreThanOneByte);

}
```

3.3 **Codice completo del software PDM**

Questo codice rappresenta il “main” del software installato sulla scheda di acquisizione del PDM. In esso sono inclusi la parte di acquisizione dati e la parte di comunicazione CAN BUS con le altre schede.

```
#include <canbus.h>
#include <PWMHandler.h>
#include <Temperature.h>
#include <Current.h>
#include <Voltage.h>
#include <RelayHandler.h>

CANBUS can;
Currents currents;
Temperatures temperatures;
Voltages voltages;
RelayHandler relay;
PWMHandler pwm;

void setup() {
  can.setupCAN();
}

void loop() {
  // Esempio di invio sul CAN di 3 dati letti da un sensore
  int dataToSend[] = { 123, 15, 230 };
  int dataLength = sizeof(dataToSend) / sizeof(dataToSend[0]);
  can.sendMessage(0x22, dataToSend, dataLength, false);

  float temperature = temperatures.getTemperatures();
  if (temperature > 50) {
    // DO SOMETHING
  } else {
    // DO SOMETHING OTHERWISE
  }

  float voltage = voltages.getVoltage(voltages.PIN_24V, voltages.RATIO_24V);
  if (voltage > 3.3) {
    // DO SOMETHING
  } else {
    // DO SOMETHING OTHERWISE
  }

  currents.readVoltages();
  currents.toCurrents();
  // DO SOMETHING WITH THE ARRAY OF CURRENTS

  // Esempio di settaggio dello stato del relay al pin 10
  bool stateFromCan = 0;
  relay.setRelayState(stateFromCan, relay.PIN10);

  // Esempio di una generazione di diversi segnali PWM
  // in base ai parametri 'dutycycle' e alla 'frequenza'
  float dutyCyclesFromCAN[] = { 10.0f, 90.0f, 40.0f, 100.0f };
  float frequenciesFromCAN[] = { 100.0f, 200.0f, 50.0f, 100.0f };
  pwm.sendPWMSignal(dutyCyclesFromCAN, frequenciesFromCAN);
}
```

Capitolo 4
TRACKING GPS

Per inseguire il *driver* di innovazione, la Polimarche ha introdotto la tecnologia GPS (Global Positioning System) all'interno del veicolo affinché sia possibile ottenere informazioni in tempo reale riguardo a:

- Tracciare graficamente il percorso della pista (ogni anno il tracciato proposto è differente).
- Calcolare il *lap time* ossia il tempo impiegato per completare un giro.
- Calcolare il *delta time* ossia lo scarto di tempo tra il giro attuale e l'ultimo completato e/o tra il giro attuale e quello migliore conseguito. Questo parametro fornisce indicazioni di come il pilota sta gareggiando.
- Comparare i diversi giri e valutare il setup di guida ottimale.

4.1 Funzionamento del GPS

Un sistema GPS è composto da tre parti:

- Satelliti: sono 31 quelli che attualmente compongono il segmento spaziale GPS. Orbitano attorno alla terra a circa 20.000 chilometri di altitudine con una velocità di 11.200 chilometri orari effettuando due rivoluzioni attorno al globo in 24 ore. Ogni satellite emette segnali che consentono ai ricevitori di determinare la loro posizione e l'ora.
- Stazioni sulla terra: usano dei radar per il tracciamento dello spostamento dei satelliti.
- Ricevitori: i classici moduli GPS integrati, per esempio, nei nostri smartphone. Questi calcolano la differenza di tempo tra l'ora di ricezione del segnale e l'ora in cui è stato inviato dal satellite per determinare la distanza tra il ricevitore e il satellite.

Il GPS utilizza un sistema, chiamato *trilaterazione*, che permette di determinare con precisione la posizione di un ricevitore sfruttando le distanze da esso a tre o più punti di riferimento noti, ovvero i satelliti. Per determinare la posizione di un oggetto, il GPS misura il tempo impiegato da un segnale radio emesso dal satellite a raggiungere l'oggetto. La velocità dell'informazione (velocità della luce) è nota, quindi misurando il tempo, è possibile calcolare la distanza tra l'oggetto e il satellite.

Per visualizzare il funzionamento della *trilaterazione*, si immagini di avere tre satelliti di cui si conoscono le posizioni nello spazio e le distanze dal ricevitore. Si ipotizzi di disegnare una sfera di centro il satellite e raggio la distanza che intercorre tra il satellite e il ricevitore GPS. Applicando il procedimento per tutti e tre i satelliti in orbita si trova l'unica intersezione tra le sfere, la quale indicherà la nostra posizione attuale. Con soli tre satelliti si riesce a trovare con precisione la latitudine e la longitudine, ma in Figura 4-1 e in Figura 4-2 viene mostrato anche un quarto

satellite. L'utilizzo di quest'ultimo permette di determinare anche l'altitudine e quindi avere una mappa tridimensionale della posizione.

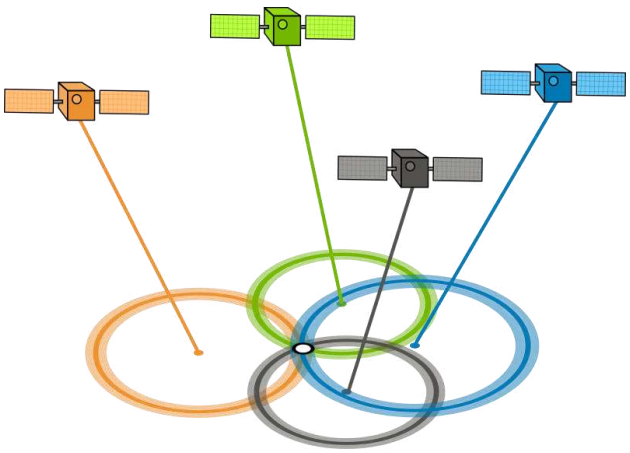


Figura 4-1. Trilaterazione 2D



Figura 4-2. Trilaterazione 3D

4.2 **Progettazione del software GPS**

Per svolgere i test di rilevamento della posizione attuale del veicolo e il tracking della pista di gara si è adoperato un *Adafruit Ultimate GPS v3*, un modulo GPS che ha una frequenza di aggiornamento massima di 10Hz (che può fornire la posizione attuale del veicolo 10 volte al secondo) ma con una tolleranza sulla precisione delle coordinate di circa tre metri. Il software mira a conoscere, ogni istante di tempo, la posizione della macchina e di fornire il numero di giri completati. Salvando tutte le coordinate è possibile graficare il tracciato di gara.

Prerequisiti per il corretto funzionamento del software:

- Il programma viene eseguito sulla linea di partenza della pista;
- Il veicolo si posiziona al centro della carreggiata.

Sulla base di questi prerequisiti il codice è stato così progettato:

- Inizialmente si acquisisce un numero di coordinate sufficienti per poter calcolare la posizione iniziale. Il numero di coordinate da leggere dipende dall'affidabilità dei dati che il sensore garantisce. In questo caso, data la scarsa precisione del sensore, si è impiegato un numero di letture di coordinate elevato (circa 50).
- Dopodiché disegnare una circonferenza di centro le coordinate appena calcolate e raggio la distanza che c'è tra il modulo GPS e la carreggiata del tracciato. Quindi si avrà una circonferenza di punti che coprono l'intera larghezza del tracciato.
- Quando la macchina sta per completare un giro, quindi è in prossimità dello *start*, si attiva la condizione che prende le coordinate all'istante t_n e le confronta con quelle acquisite all'istante t_0 , verificando se si trova all'interno della circonferenza. Se sì, allora incrementa il contatore del numero di giri completati.

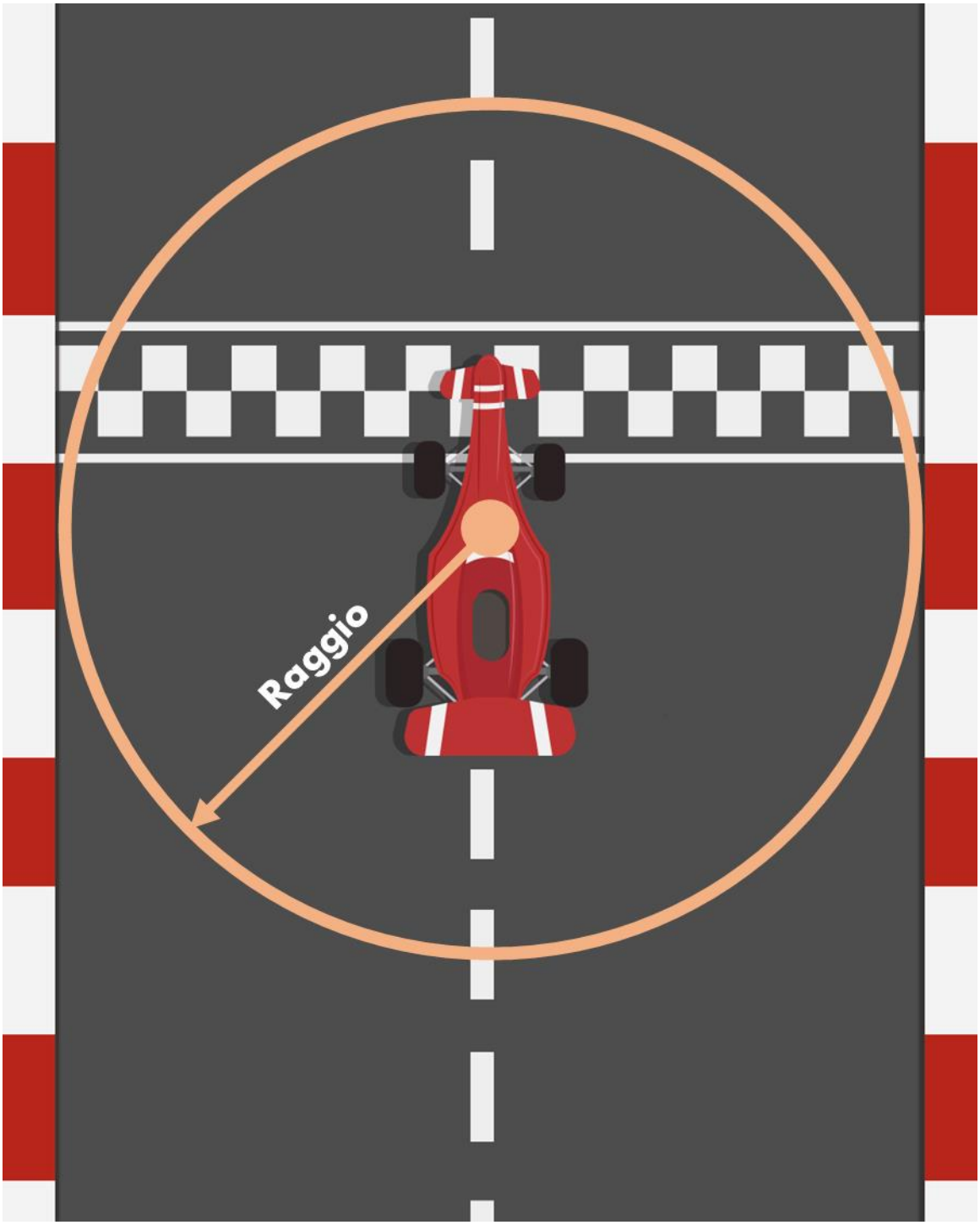


Figura 4-3. Circonferenza di punti attorno alla coordinata iniziale

4.3 **Testing su MATLAB**

Per il testing del software, è stato simulato un ipotetico tracciato circolare utilizzando un veicolo e il modulo GPS integrato su di esso. I dati raccolti sono stati successivamente riportati su MATLAB e analizzati. La Figura 4-4 mostra il percorso simulato prendendo le coordinate direttamente da Google Earth con un apposito strumento software, mentre la Figura 4-5 visualizza il risultato ottenuto unendo i punti letti dal GPS tramite dei segmenti.

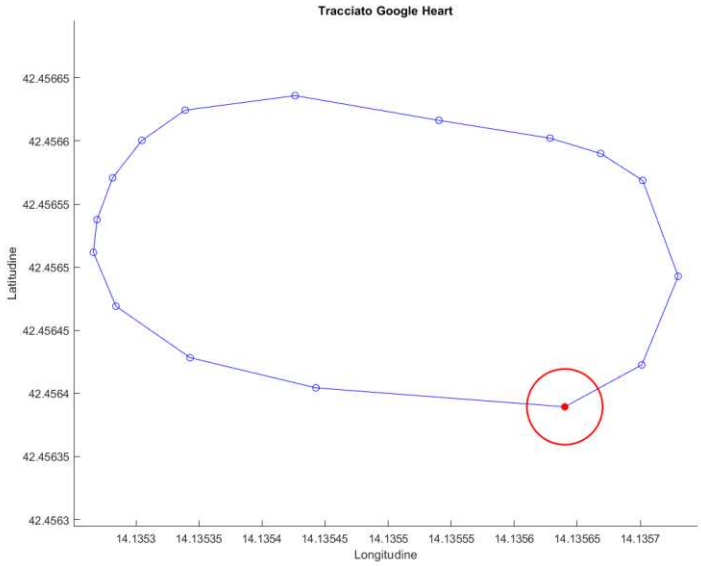


Figura 4-4. Tracciato simulato

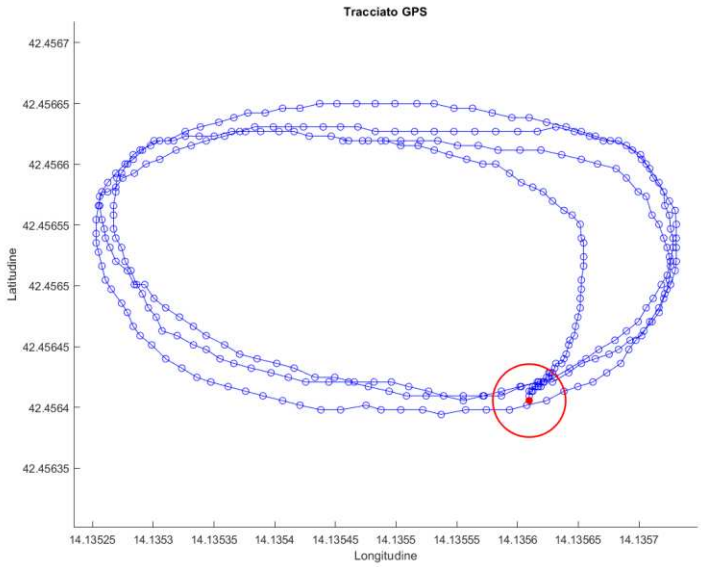


Figura 4-5. Tracciato coordinate GPS

4.3.1 *Codice MATLAB*

GPS_Google_Earth.m

```
% Coordinate dei punti
coordinates = [
    14.13564008290164,42.45638933853132;
    14.13570114422611,42.45642258583789;
    14.13573004544705,42.4564927138641;
    14.13570187482711,42.45656869702832;
    14.13566858160563,42.45659002344726;
    14.13562833637383,42.45660216105944;
    14.13554025625185,42.45661625411417;
    14.13542624933452,42.45663587242306;
    14.1353389921849,42.45662436418704;
    14.1353046230462,42.45660033370964;
    14.13528144108153,42.45657074451086;
    14.13526908406548,42.45653758082745;
    14.13526617568013,42.45651170990243;
    14.13528413715374,42.45646900996181;
    14.13534286733688,42.45642829130698;
    14.13544269016338,42.4564043867596;
    14.13564008290164,42.45638933853132;

];

% Chiamata alla funzione per disegnare il percorso chiuso
plotPath(coordinates);

function plotPath(coordinates)
    % Assumiamo che la prima coordinata sia quella iniziale
    coordinates = [coordinates; coordinates(1,:)];

    hold on;

    % Disegna le rette che connettono le coordinate
    plot(coordinates(:,1), coordinates(:,2), 'b-o');

    % Segna il punto di partenza con un pallino rosso pieno
    plot(coordinates(1,1), coordinates(1,2), 'ro', 'MarkerFaceColor', 'r');

    % Crea una circonferenza attorno al punto di partenza
    radius = 0.00003;
    center = coordinates(1,:);
    theta = linspace(0, 2*pi, 100);
    x = center(1) + radius * cos(theta);
    y = center(2) + radius * sin(theta);
    plot(x, y, 'r', 'LineWidth', 1.5);

    % Impostazioni del grafico
    xlabel('Longitudine');
    ylabel('Latitudine');
    title('Tracciato Google Heart');
    axis equal;
end
```

GPS_Tracking.m

```
% Imposta il nome del file .csv da caricare
% Nel file sono salvate le coordinate lette dal GPS
file_name = 'coordinate100ms.csv';

% Carica i dati dal file .csv
coordinates = csvread(file_name);
coordinates = coordinates(:, [2, 1]);

% Chiamata alla funzione per disegnare il percorso chiuso
plotPath(coordinates);

function plotPath(coordinates)
    % Assumiamo che la prima coordinata sia quella iniziale
    coordinates = [coordinates; coordinates(1,:)];

    hold on;

    % Disegna le rette che connettono le coordinate
    plot(coordinates(:,1), coordinates(:,2), 'b-o');

    % Segna il punto di partenza con un pallino rosso pieno
    plot(coordinates(1,1), coordinates(1,2), 'ro', 'MarkerFaceColor', 'r');

    % Crea una circonferenza attorno al punto di partenza
    radius = 0.00003;
    center = coordinates(1,:);
    theta = linspace(0, 2*pi, 100);
    x = center(1) + radius * sin(theta);
    y = center(2) + radius * cos(theta);
    plot(x, y, 'r', 'LineWidth', 1.5);


    % Impostazioni del grafico
    xlabel('Longitudine');
    ylabel('Latitudine');
    title('Tracciato GPS');
    axis equal;
end
```

4.4 **Codice completo del software GPS**

```
#include <algorithm>
#include <Adafruit_GPS.h>
#include <math.h>

// Settings GPS
#define GPSSerial Serial1
#define GPSECHO false
Adafruit_GPS GPS(&GPSSerial);

// Settings Parameters
#define Raggio_terrestre 6371000.0f
uint32_t timer = millis();
uint8_t attemptToFindCenter = 0;                    // Indice che monitora il numero di
acquisizioni per trovare la coordinata da cui partire
const uint8_t numCoordinates = 50;                  // Numero di coordinate iniziali da
cui trovare la mediana
const float minVelocityKmh = 5.0f;                  // (Km/h) Velocità minima per
determinare che la macchina è in movimento
const float minVelocityMs = minVelocityKmh / 3.6f;  // (m/s) Velocità minima per
determinare che la macchina è in movimento
float arrayOfCoords[numCoordinates][2];             // Matrice di coordinate ausiliaria
che memorizza "numCoordinates" coordinate
const float toMS = 1.852f;                          // Parametro che serve per convertire
la velocità da nodi a metri al secondo
const float radius = 7.0f;

// Variables
float actualLat = 0.0f;       // Latitudine attuale
float actualLon = 0.0f;       // Longitudine attuale
float deltaLat = 0.0f;        // Distanza latitudinale
float deltaLon = 0.0f;        // Distanza longitudinale
float speed = 0.0f;           // Velocità attuale
bool isRacing = false;        // La macchina sta gareggiando? Ha una velocità superiore a
"minVelocityMs"?
bool isInsideCircle = false;  // La macchina è all'interno del reaggio del cerchio?
uint8_t lapCompleted = 0;     // Numero di giri completati
bool enable = true;           // Serve come controllo aggiuntivo al conteggio dei giri.
Fa si che aggiorni il contatore di solo 1 unità per giro
bool isFirstLap = true;       // Serve come controllo aggiuntivo al conteggio dei giri.
Fa si che non conti la partenza come un giro completato

// Data Structure
struct Circle {
  float centerX;
  float centerY;
  float radius;
};

Circle circle;  // Definisce la struttura dati Circle che disegna la circonferenza
intorno alla coordinata di partenza

void setup() {
  Serial.begin(115200);
  GPS.begin(9600);
  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_10HZ);
  GPS.sendCommand(PMTK_API_SET_FIX_CTL_5HZ);
  GPS.sendCommand(PGCMD_ANTENNA);
  GPS.sendCommand(PMTK_ENABLE_WAAS);
  delay(1000);
```

```
  circle.radius = radius;
}

void loop()
{
  char c = GPS.read();
  if (GPS.newNMEAreceived()) {
    //Serial.print(GPS.lastNMEA());           // Se decommentata, stampa su seriale la
stringa NMEA
    if (!GPS.parse(GPS.lastNMEA()))
      return;
  }

  // Aggiornamenti ogni 100 millisecondi
  if (millis() - timer > 100) {
    timer = millis();
    if (GPS.fix) {
      if (attemptToFindCenter <= numCoordinates) {
        setInitCoordinates(circle);
      } else {
        actualLat = GPS.latitudeDegrees;
        actualLon = GPS.longitudeDegrees;
        speed = GPS.speed * toMS;

        isInsideCircle = areCoordsInsideCircle(actualLat, actualLon, circle);
        isRacing = isInMovement(speed);

        if (isRacing) {
          if (isInsideCircle) {
            if (enable && !isFirstLap) {
              lapCompleted++;
              enable = false;
            }
          } else {
            enable = true;
            isFirstLap = false;
          }
        }

        printInformations();

        Serial.println();
      }
    }
  }
}

void setInitCoordinates(Circle& circle) {
  if (attemptToFindCenter == numCoordinates) {
    circle.centerX = medianLon(arrayOfCoords);
    circle.centerY = medianLat(arrayOfCoords);
    attemptToFindCenter++;
  } else {
    arrayOfCoords[attemptToFindCenter][0] = GPS.latitudeDegrees;
    arrayOfCoords[attemptToFindCenter][1] = GPS.longitudeDegrees;
    attemptToFindCenter++;
  }
}

// Funzione per calcolare la mediana delle latitudini
float medianLat(float coords[numCoordinates][2]) {
  float latitudes[numCoordinates];
  for (int i = 0; i < numCoordinates; i++) {
    latitudes[i] = coords[i][0];
```

```
  }
  // Ordina l'array delle latitudini
  std::sort(latitudes, latitudes + numCoordinates);

  // Calcola la mediana delle latitudini
  if (numCoordinates % 2 == 0) {
    return (latitudes[numCoordinates / 2 - 1] + latitudes[numCoordinates / 2]) / 2.0f;
  } else {
    return latitudes[numCoordinates / 2];
  }
}

// Funzione per calcolare la mediana delle longitudini
float medianLon(float coords[numCoordinates][2]) {
  float longitudes[numCoordinates];
  for (int i = 0; i < numCoordinates; i++) {
    longitudes[i] = coords[i][1];
  }
  // Ordina l'array delle longitudini
  std::sort(longitudes, longitudes + numCoordinates);

  // Calcola la mediana delle longitudini
  if (numCoordinates % 2 == 0) {
    return (longitudes[numCoordinates / 2 - 1] + longitudes[numCoordinates / 2]) / 2.0f;
  } else {
    return longitudes[numCoordinates / 2];
  }
}

float toRadians(float degree) {
  return degree * M_PI / 180.0;
}

/**
  Funzione per verificare se una coordinata si trova all'interno o all'esterno della
circonferenza
  Calcola la distanza longitudinale e latitudinale tra il centro della circonferenza e la
coordinata passata come parametro
  e verifica che le distanze sono all'interno del raggio
**/
bool areCoordsInsideCircle(float lat, float lon, Circle circle) {
  deltaLat = fabs(latitudinalDistance(circle.centerY, lat));
  deltaLon = fabs(longitudinalDistance(circle.centerX, lon, circle.centerY));

  return (deltaLat <= circle.radius) && (deltaLon <= circle.radius);
}

// Funzione per calcolare la distanza longitudinale tra due punti sulla superficie della
Terra
float longitudinalDistance(float lon1, float lon2, float lat) {
  return toRadians(lon2 - lon1) * Raggio_terrestre * cos(toRadians(lat));
}

// Funzione per calcolare la distanza latitudinale tra due punti sulla superficie della
Terra
float latitudinalDistance(float lat1, float lat2) {
  return toRadians(lat2 - lat1) * Raggio_terrestre;
}

bool isInMovement(float speed) {
  if (speed <= minVelocityMs) {
    return false;
  }
  return true;
```

```
}

void printInformations() {
  printTime();
  Serial.println();
  Serial.println("Posizione attuale:      " + String(actualLon, 7) + " : " +
String(actualLat, 7));
  Serial.println("Centro del cerchio:     " + String(circle.centerX, 7) + " : " +
String(circle.centerY, 7));
  Serial.println("Velocità (m/s):         " + String(speed));
  Serial.println("Distanza longitudinale: " + String(deltaLon));
  Serial.println("Distanza latitudinale:  " + String(deltaLat));

  if (lapCompleted != 0) {
    Serial.println("--->Ha completato " + String(lapCompleted) + " giri.");
  }

  if (isInsideCircle) {
    Serial.println("------>Dentro il cerchio.");
  } else {
    Serial.println("------>Fuori dal cerchio.");
  }

  if (isRacing) {
    if (isInsideCircle) {
      Serial.println("------>GIRO COMPLETATO.");
    } else {
      Serial.println("------>GIRO NON COMPLETATO.");
    }
  } else {
    Serial.println("------>Veicolo fermo.");
  }
}
```

Capitolo 5
SITOGRAFIA

- Formula SAE storia: https://www.formula-ata.it/
- Formula SAE gare: https://www.formula-ata.it/formula-sae-italy/
- Equazione di Steinhart-Hart: https://it.wikipedia.org/wiki/Equazione_di_Steinhart-Hart
- CAN BUS: https://www.researchgate.net/publication/353499375_An_Overview_of_CAN-BUS_Development_Utilization_and_Future_Potential_in_Serial_Network_Messaging_for_Off-Road_Mobile_Equipment
- GPS:
    - https://spaceplace.nasa.gov/gps/en/#:~:text=The%20Short%20Answer%3A,figure%20out%20where%20you%20are.
    - https://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/navservices/gnss/gps/howitworks
- Trilaterazione: https://gisgeography.com/trilateration-triangulation-gps/

Capitolo 6
RINGRAZIAMENTI

Ringrazio il Prof. Ippoliti Gianluca per il suo sostegno durante il mio tirocinio e nella stesura della tesi.
Ringrazio i compagni del team del Polimarche. Grazie a loro, ho avuto l'opportunità di partecipare a un'esperienza straordinaria e di appassionarmi al progetto.

Ringrazio profondamente Maria e Franco, i miei genitori, insieme a mia sorella Federica. Sono stati il mio costante punto di riferimento in tutti questi anni, rappresentando il rifugio sicuro in cui ho trovato conforto e supporto in ogni momento. La mia famiglia ha contribuito a rafforzare la mia autostima e a spronarmi costantemente a cercare di migliorare, diventando una persona migliore.

Ringrazio i miei parenti, fondamentali nel mio percorso accademico e personale. Ognuno di loro, con le proprie qualità uniche, ha contribuito in modo significativo a forgiare il mio carattere. La loro saggezza, amore incondizionato e costante presenza hanno arricchito la mia crescita, insegnandomi importanti lezioni di vita. La determinazione incrollabile, la gentilezza disarmante e la capacità di essere comprensivi e di pronunciare le parole giuste al momento giusto, mi hanno ispirato a diventare una persona più compassionevole, resiliente e capace di ascoltare. La famiglia è stata per me una scuola di valori, e sono grato per l'enorme influenza positiva che ha avuto sulla mia vita.

Ringrazio i miei migliori amici Chiara, Giulia e Niccolò per essere stati al mio fianco in ogni momento. Insieme abbiamo costruito un legame unico basato sulla fiducia reciproca e sull'empatia, sostenendoci a vicenda nelle sfide della vita. Abbiamo condiviso momenti di crescita personale, ispirandoci a vicenda a diventare persone migliori. Non sono mancati i momenti di pura gioia e divertimento che hanno cementato ancora di più la nostra amicizia.

Ringrazio il mio fedele compagno Davide, il mio confidente più fidato, la persona con la quale ho condiviso innumerevoli momenti meravigliosi. Grazie a lui, i giorni non sono mai stati banali o monotoni, ma sempre pieni di emozioni e avventure. Sono stato fortunato ad averlo conosciuto, dato che condivide il mio stesso senso dell'umorismo, facendomi ridere in ogni situazione e aggiungendo un tocco speciale di allegria alla nostra amicizia.

Ringrazio tutti i miei compagni di Pescara per la loro calorosità e amicizia. Le vostre presenze allietano le mie giornate nella città che amo: Lollocik, Montich, Luca, Sbrattino, Angy, Pasqualucci, Nik, Escanor, Leonelli, Giorgia, Isabella, Francisca, Riti, Greta, Mattia, Chiara, Josè, Merla, Sasha, Cillo, Jordan, Ada.

Ringrazio i miei amici di Ancona, con i quali ho condiviso questi anni indimenticabili: Cecilia, Camilla, Benedetta, Roberto, Simone Di Battista, Simone Amato, Gianluca, Matteo, Tommaso, Sofia, Giada, Giulia, Ilenia.