UNIVERSITÀ
POLITECNICA
DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea in Ingegneria Meccanica

Studio e implementazione di un sistema a microcontrollore per l'acquisizione di parametri di diagnostica del veicolo su BUS CAN

Study and development of a microcontroller-based acquisition system of vehicle diagnostic parameters on CAN BUS interface

Tesi di laurea di:

Simona Piemontese

Relatore:

Prof. **Alessandro Terenzi**

Correlatori

Dott. Ing. **Valeria Bruschi**
Prof. **Gianluca Ciattaglia**

Anno Accademico 2023-2024

A nonno Franco,
che continua a vivere al mio fianco,
che più di chiunque altro
avrebbe voluto esserci.
Grazie per avermi ispirato e per essere stata
la mia guida anche oltre le stelle.

A mia mamma e a mio padre,
luce dei miei occhi,
la forza più pura che io conosca.

A mia sorella Sara,
il mio sole,
metà del mio cuore e della mia anima.

Abstract

Il presente lavoro di tesi si propone di analizzare e implementare un sistema a microcontrollore per l'acquisizione di parametri diagnostici di veicoli attraverso la rete di comunicazione CAN (Controller Area Network).
L'obiettivo principale è quello di sviluppare una soluzione agevole per il monitoraggio delle prestazioni in modo da verificare la sicurezza del veicolo e migliorarne l'affidabilità.
La diagnostica di un veicolo si basa sull'analisi di una serie di dati trasmessi tramite il BUS CAN, il quale rappresenta un mezzo di collegamento tra le varie centraline elettroniche del veicolo.

Il lavoro svolto si articola in diverse fasi. La prima parte del progetto è stata dedicata alla revisione della letteratura riguardante il protocollo CAN e le applicazioni nel settore automotive.
La seconda parte del progetto è stata destinata all'utilizzo di Arduino e di vari dispositivi ad esso collegati per l'acquisizione dei messaggi su protocollo CAN.
Il lavoro ha riguardato un approfondito studio delle funzionalità della scheda Arduino, concentrandosi in particolare sull'utilizzo di diverse librerie e sulla scrittura di codici specifici per le applicazioni desiderate.
Sono stati impiegati i seguenti dispositivi: Arduino R3 UNO e BUS CAN shield. Sono stati inoltre utilizzati jumpers e un simulatore a pacchetti.

Lo scopo finale del progetto nel quale si inserisce questa tesi è quello di realizzare un sistema di monitoraggio per la raccolta e l'analisi dei dati provenienti da sensori vari in tempo reale.

Abstract

The present thesis work aims to analyze and implement a microcontroller-based system for acquiring diagnostic parameters of vehicles through the CAN (Controller Area Network) communication network.
The main objective is to develop an easy-to-use solution for performance monitoring to verify vehicle safety and improve its reliability.
The diagnostics of a vehicle are based on the analysis of a series of data transmitted via the CAN BUS, which serves as a means of connecting the various electronic control units in the vehicle.

The work is structured in several phases. The first part of the project was dedicated to the literature review regarding the CAN protocol and its applications in the automotive sector.
The second part of the project focused on the use of Arduino and various connected devices for acquiring messages on the CAN protocol.
The work involved an in-depth study of the functionalities of the Arduino board, particularly concentrating on utilizing different libraries and writing specific codes for the desired applications. The following devices were used: Arduino R3 UNO and CAN BUS shields. Additionally, jumpers and a packet simulator were employed.

The ultimate goal of the project, in which this thesis is situated, is to create a monitoring system for the collection and analysis of data from various sensors in real time.

Indice

Introduzione

Nel corso degli ultimi anni l'evoluzione del settore automotive è stata orientata nella ricerca di soluzioni innovative per migliorare la sicurezza e l'affidabilità dei veicoli.

A tal proposito, la diagnostica si presenta come metodo efficace di valutazione in quanto permette di analizzare le prestazioni del veicolo ottimizzando gli interventi di manutenzione.

Il presente lavoro di tesi si colloca all'interno di questo panorama in continuo sviluppo, concentrandosi sulla comunicazione BUS CAN attraverso una scheda microcontrollore Arduino.

La linea BUS CAN costituisce una soluzione versatile e accessibile per consentire il trasferimento di dati in tempo reale tra le centraline e i sensori del veicolo.

La scheda Arduino è una scheda elettronica programmabile ed è stata il dispositivo utilizzato per l'acquisizione dati.

La prima fase del progetto è stata dedicata allo studio della letteratura esistente sul protocollo CAN e il suo continuo sviluppo nel settore automotive.

Successivamente, il lavoro si è concentrato nell'acquisizione dati utilizzando vari dispositivi connessi alla scheda Arduino, quali Arduino R3 UNO e BUS CAN shield.

L'impiego del jumper per collegare un pin analogico al pin 5V è stato essenziale poiché ha consentito, grazie a un codice sviluppato su Arduino, di leggere il valore della tensione.

È stato effettuato un tentativo di lettura dei dati provenienti da un simulatore a pacchetti. Questa operazione è stata condotta utilizzando un esempio

specifico di una libreria scaricata su Arduino. L'obiettivo di questo tentativo è stato quello di acquisire e analizzare le informazioni trasmesse dal simulatore.

Inoltre, è stata eseguita la lettura delle posizioni del joystick integrato sulla BUS CAN shield. I dati raccolti, provenienti dalle variazioni di posizione del joystick, sono stati poi archiviati su una scheda microSD inserita all'interno dello BUS CAN shield, consentendo la raccolta di input direttamente da un dispositivo di controllo di tipo analogico.

Successivamente, è stata effettuata una connessione tra due BUS CAN shield tramite l'uso di jumpers. In questo contesto i due dispositivi si identificano uno come *master* e l'altro come *slave*.

Il *master* ha il compito di gestire e controllare il flusso di informazioni all'interno della rete, mentre lo *slave* si occupa di ricevere e rispondere ai comandi impartiti dal master.

Capitolo 1: Protocollo CAN

Il CAN BUS (Controller Area Network) [1] è un protocollo basato su messaggi progettato per consentire alle unità di controllo elettronico (ECU) presenti nelle automobili di comunicare tra loro in modo affidabile ed è basato sulle priorità.

Si tratta di uno standard ISO (ISO 11898) che consente un controllo in tempo reale con un livello di sicurezza molto elevato.

Dopo l'embargo petrolifero degli anni '70, le case automobilistiche subirono una crescente spinta a ottimizzare l'efficienza dei consumi di carburante.

Di conseguenza, iniziarono a esplorare strategie per ridurre il peso dei veicoli in produzione.

All'inizio degli anni '80, le auto avevano sempre più unità di controllo elettronico al loro interno e aziende come la tedesca Robert Bosch erano alla ricerca di un tipo di sistema di comunicazione BUS che potesse essere utilizzato come sistema di comunicazione tra più ECU e sistemi del veicolo.

Nel 1986, Bosch introdusse il protocollo CAN per applicazioni automobilistiche al Congresso SAE di Detroit, per poi diffondersi in molti settori dell'industria.

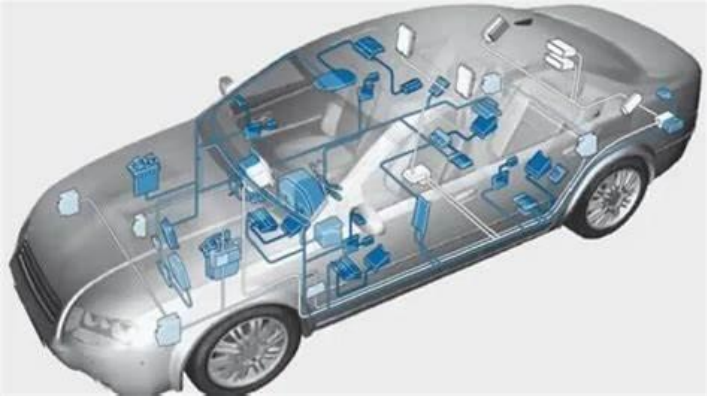


Figura 1. Rete CAN in un'automobile

1.1 Applicazioni più diffuse del protocollo CAN

Il protocollo CAN è un sistema di comunicazione seriale robusto e altamente affidabile, originariamente sviluppato per il settore automobilistico, che ha saputo guadagnarsi una posizione di leadership anche in numerosi altri ambiti applicativi [2]. La sua capacità di operare in condizioni critiche, caratterizzate da elevati livelli di interferenza elettromagnetica e temperature estreme, lo rende una scelta preferita per molteplici settori.

Nel settore automotive il protocollo CAN è utilizzato praticamente in ogni tipo di veicolo, che siano automobili, camion o altri mezzi di trasporto.
Le case automobilistiche di prestigio, come Audi, BMW, Renault e Volkswagen, integrano reti CAN nei loro modelli per garantire una comunicazione efficiente e sicura tra i diversi sistemi elettronici del veicolo.
Oltre all'industria automobilistica, l'applicazione del protocollo CAN si estende all'elettronica marittima, dove è impiegato per gestire sistemi di navigazione, comunicazione e controllo a bordo di navi e imbarcazioni diverse.
Questo protocollo si dimostra estremamente utile anche nel settore aeronautico e aerospaziale, dove contribuisce a garantire la sicurezza e l'affidabilità dei sistemi di volo e dei vari dispositivi elettronici presenti sugli aerei.
Un altro campo rilevante in cui il protocollo CAN trova applicazione è l'automazione di fabbrica. Qui, il protocollo permette la comunicazione tra macchine, robot e sistemi di controllo, ottimizzando i processi produttivi e migliorando l'efficienza operativa. Parallelamente, nel contesto del controllo di macchine industriali, il protocollo garantisce che i dispositivi e i sensori

possano scambiare informazioni in tempo reale, facilitando la supervisione e la gestione degli impianti industriali.

Infrastrutture urbane come ascensori e scale mobili beneficiano anch'esse dell'implementazione del protocollo CAN, che assicura un controllo preciso e sicuro delle operazioni. Inoltre, il protocollo viene utilizzato nell'automazione degli edifici per gestire sistemi di sicurezza, illuminazione e climatizzazione, contribuendo a creare ambienti più intelligenti ed efficienti.

Infine, il protocollo CAN è applicato anche nel settore biomedicale, dove supporta la comunicazione tra dispositivi medici e componenti elettronici, garantendo il monitoraggio e la gestione sicura delle informazioni critiche. La versatilità del protocollo CAN si estende anche al controllo e ai dispositivi non industriali, dimostrando la sua utilità in un'ampia gamma di contesti e applicazioni.

1.2 Vantaggi del protocollo CAN

Il protocollo CAN è considerato il futuro standard predominante nelle reti industriali. Questo è principalmente attribuibile ai seguenti benefici [3]:

1. **tempi di risposta rigidi**, specifica fondamentale nel controllo di processo: la tecnologia CAN prevede molti strumenti hardware e software e sistemi di sviluppo per protocolli ad alto livello che consentono di connettere un elevato numero di dispositivi mantenendo stringenti vincoli temporali;
2. **semplicità e flessibilità del cablaggio**: il CAN è un bus seriale tipicamente implementato su un doppino intrecciato (schermato o meno a seconda delle esigenze). I nodi non hanno un indirizzo che li

identifichi e possono quindi essere aggiunti o rimossi senza dover riorganizzare il sistema;

3. **alta immunità ai disturbi**: lo standard ISO 11898 garantisce che i chips di interfaccia possano continuare a comunicare anche in condizioni estreme, come l'interruzione di uno dei due fili o il cortocircuito di uno di essi con massa o con l'alimentazione;
4. **elevata affidabilità**: la rilevazione degli errori e la richiesta di ritrasmissione viene gestita direttamente dall'hardware con cinque diversi metodi (due a livello di bit e tre a livello di messaggio).

   Il bus can presenta un'elevata capacità di rilevamento e correzione degli errori.

   Infatti, la probabilità che un messaggio sia corrotto e non riconosciuto come tale, è praticamente nulla.

   È stato calcolato che una rete basata su CAN BUS a 1 Mbit/s, con un utilizzo medio del bus del 50%, una lunghezza media dei messaggi di 80 bit e un tempo di lavorazione di 8 ore al giorno per 365 giorni l'anno, avrà un errore non rilevato ogni 1000 anni, quindi la rete non è soggetta ad errori per tutta la durata della sua vita;
5. **confinamento degli errori**: ciascun nodo è in grado di rilevare il proprio malfunzionamento e di autoescludersi dal bus se questo è permanente. Questo è uno dei meccanismi che consentono alla tecnologia CAN di mantenere la rigidità delle temporizzazioni, impedendo che un solo nodo metta in crisi l'intero sistema;
6. **maturità dello standard**: la larga diffusione del protocollo CAN ha determinato un'ampia disponibilità di chip rice-trasmettitori, di microcontrollori che integrano porte CAN, di tools di sviluppo, oltre che una sensibile diminuzione del costo di questi sistemi. Questo è

fondamentale affinché uno standard riesca a consolidarsi nel settore industriale.

1.3 Modalità broadcast e tipi di messaggio

I dispositivi su un BUS CAN sono chiamati "nodi".

Ogni nodo è costituito da una CPU, un controller CAN e un ricetrasmettitore, che adatta i livelli del segnale sia dei dati inviati che ricevuti dal nodo.

Tutti i nodi sono in ascolto e recepiscono tutti i messaggi, ma non si può spedire un messaggio a un nodo specifico.

Non è previsto un campo di indirizzamento; l'indirizzo può essere implicito nel contenuto del messaggio trasmesso.

I nodi sono sincronizzati in modo che campionano i dati sulla rete contemporaneamente. Tuttavia, i dati non vengono trasmessi con i segnali di clock, quindi il CAN non è veramente un bus sincrono.

Nel protocollo CAN esistono cinque differenti strutture di messaggi:

- **Data Frame**: è [3] il tipo di messaggio più diffuso e permette la trasmissione dei dati da un nodo trasmettitore (TX) a tutti gli altri, che si comportano quindi come ricevitori (RX).

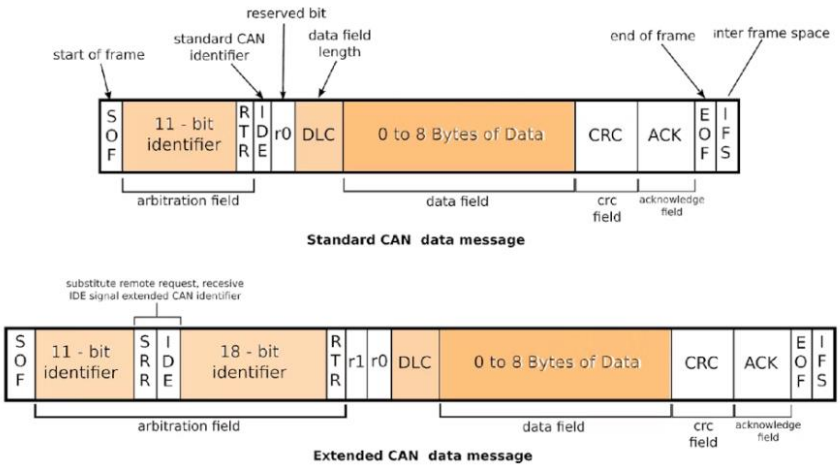


Figura 2. Standard CAN e Extended CAN

È costituito dai seguenti campi [2]:

- **Arbitration Field**: determina la priorità del messaggio in caso di contesa e contiene:
  - Per CAN 2.0A (Standard CAN), un identificatore 11-bit e un Remote Transmission Request (RTR) bit, che è dominante per i data frames.
  - Per CAN 2.0B (Extended CAN), un identificatore 29-bit e un RTR bit l.
- **Control Field**: è costituito da 6 bit, di cui 4 servono a specificare la lunghezza del Data Field e 2 sono riservati per future espansioni del protocollo.
- **Data Field**: contiene da zero a 8 bytes di dati.
- **CRC** (Cyclic Redundancy Check) **Field**: contiene 15-bit di verifica dell'errore sul messaggio

- **Remote Frame**: ha [2] una struttura simile al Data Frame, ma è privo del campo dati e ha un RTR recessivo. Serve per sollecitare l'invio di un dato da parte di un data frame con lo stesso frame con lo stesso identificativo.

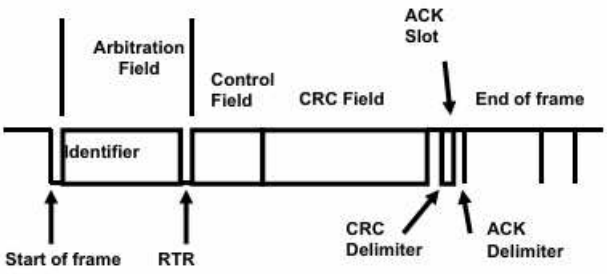


Figura 3. Remote Frame

- **Error Frame**: viene inviato da un nodo che rivela un errore e provoca la ritrasmissione del messaggio da parte del nodo trasmettitore.
  Poiché è sufficiente che un solo nodo segnali un errore per avere la ritrasmissione, il protocollo CAN prevede che ciascun nodo monitorizzi il proprio stato, autoescludendosi in caso di tasso di errore elevato (*fault confinement*).

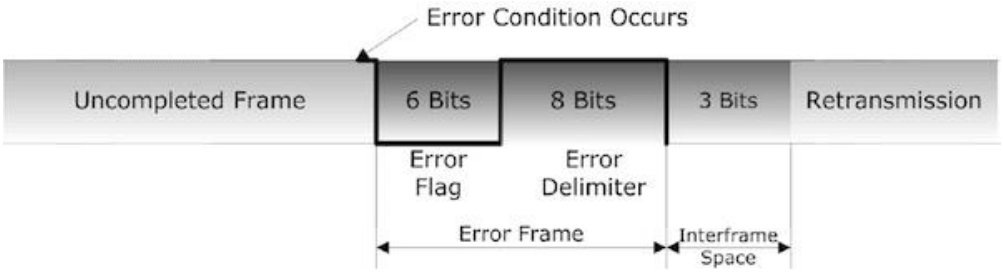


Figura 4. Error Frame

È composto da 2 campi:

  - **Error Flag**: è inviato dal nodo che rileva un errore e comprende 6 bit dello stesso livello di tipo dominante o recessivo a seconda bit dello stesso livello di tipo dominante o recessivo a seconda dell'effetto voluto sul traffico. Se i bit sono dominanti bloccano il traffico e impediscono al messaggio errato di giungere a destinazione. Possono inviare Error Flag dominanti solo nodi "affidabili", cioè che non risultano statisticamente responsabili di un numero elevato di errori.
  - **Error Delimiter**: comprende 8 bit di valore recessivo.

- **Overload Frame**: Serve [2] a ritardare l'invio di dati successivi nel caso che un nodo sia impegnato da elaborazioni precedenti e non possa gestire altri dati.

È costituito da 6 bit dominanti (che interrompono il traffico) seguiti da 8 bit recessivi.

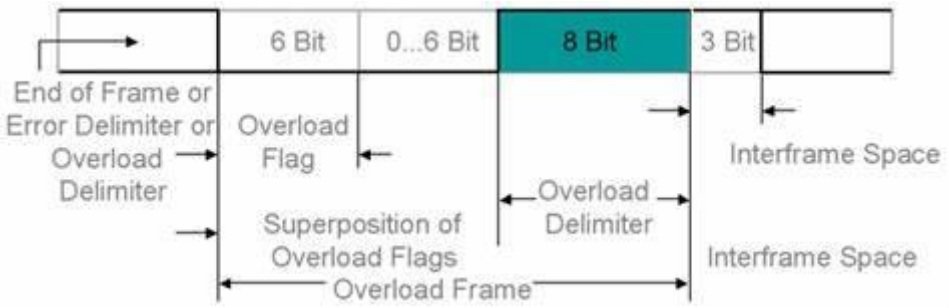


Figura 5. Overload Frame

- **Interframe Space**: precede [2] ogni Data e Remote Frame e ha una funzione separatrice.

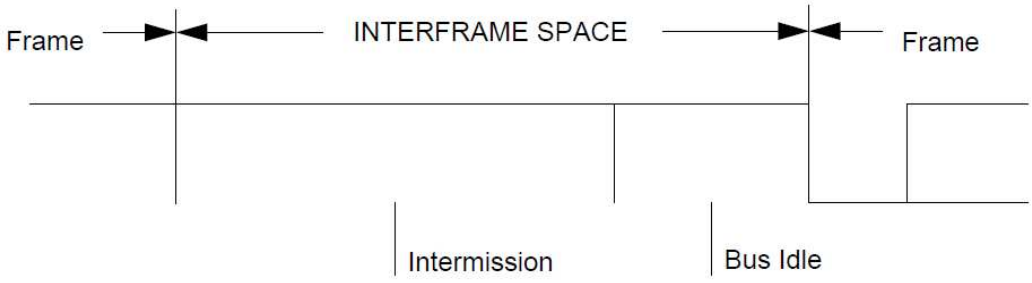


Figura 6. Interframe Space

1.4 Varianti del protocollo CAN

Lo standard ISO 11898 definisce diverse versioni del protocollo CAN.

I principali tipi di CAN utilizzati nel settore automotive sono [1]:

- **Low Speed CAN**: è utilizzato per sistemi con tolleranza ai guasti che non richiedono velocità di aggiornamento elevate.

  La velocità massima di trasferimento dati è di 125 kbps, ma il cablaggio è più economico rispetto al High Speed CAN.

  Nelle applicazioni automobilistiche, viene utilizzato, ad esempio, per la diagnostica, i controlli e i display del cruscotto, gli alzacristalli elettrici.

- **High Speed CAN**: è utilizzato per le comunicazioni tra sottosistemi critici che richiedono elevate velocità di aggiornamento e un'elevata precisione dei dati. Ne sono esempi il sistema di antibloccaggio della frenata, il controllo elettronico della stabilità, l'airbag, le unità di controllo del motore.

  Le velocità di trasferimento dati vanno da 1 kbit a 1 Mbit al secondo.

  È più veloce del Low Speed CAN, ma il requisito di larghezza di banda delle nuove applicazioni automobilistiche aumenta ogni anno, quindi gli OEM automobilistici stanno installando CAN FD nelle nuove auto.

- **CAN FD** (Flexible Data Rate CAN): è caratterizzato da una velocità di trasmissione dati flessibile, più dati per messaggio e trasmissioni molto più veloci. La velocità massima di trasmissione dati è stata aumentata rispetto ai precedenti da 1 Mbps a 8 Mbps.

Consente alle centraline di modificare dinamicamente le loro velocità di trasmissione e selezionare dimensioni dei messaggi più grandi o più piccole in base ai requisiti in tempo reale. Al momento si trova solo nei veicoli ad alte prestazioni.

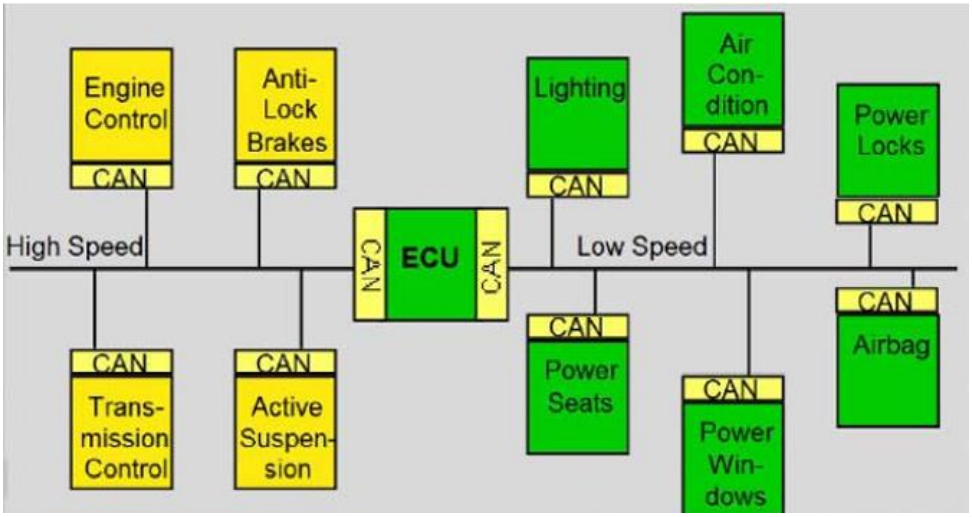


Figura 7. Rete CAN su un'automobile

CAPITOLO 2: Arduino e hardware utilizzato

2.1 Arduino

Arduino è [4] una scheda programmabile dotata di microcontrollore.

È una piattaforma elettronica open-source che unisce hardware e software per la creazione di progetti elettronici interattivi.

Si compone di una scheda di sviluppo (microcontrollore) e un ambiente di sviluppo integrato (IDE) che consente di scrivere, compilare e caricare codice.

Nasce nel 2005 all'interno dell'Interaction Design Institute di Ivrea come strumento semplice per la prototipazione veloce.

Arduino è fondamentalmente costituito da un microcontrollore montato su una scheda dotata di pin collegati alle porte di input/output, un regolatore di tensione e un'interfaccia USB che facilita la comunicazione con il computer.

2.2 Hardware utilizzato

Esistono numerose varianti della scheda Arduino, ognuna delle quali è progettata per rispondere a esigenze specifiche. Queste schede, pur condividendo una base comune, si differenziano per caratteristiche tecniche, forme, capacità di elaborazione e funzionalità.

Per lo sviluppo di questa tesi sono state utilizzate due diverse schede: Arduino R3 UNO e il BUS CAN shield.

2.2.1 Arduino R3 UNO

Si tratta di una scheda di base. È una scheda microcontrollore [5] basata su ATmega328. Dispone di 14 pin di ingresso/uscita digitali (di cui 6 utilizzabili come uscite PWM), 6 ingressi analogici, un oscillatore a cristallo da 16 MHz, una connessione USB, una presa per spinotto di alimentazione, un'intestazione ICSP e un pulsante di ripristino. L'alimentazione della scheda è data da un cavo USB connesso al computer tramite la rispettiva porta.

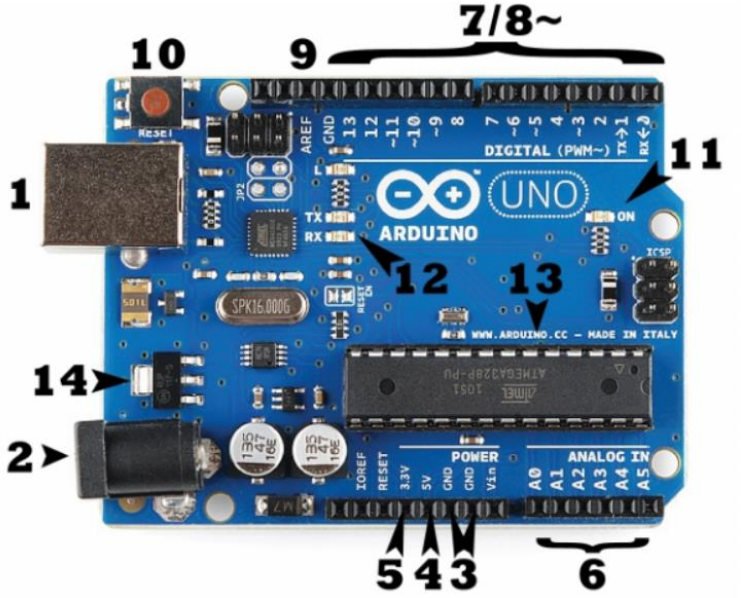


Figura 8. Arduino UNO

Arduino [6] dispone di diversi pin tra cui:

- Il pin 3,3 V, 5 V e GND (3,4,5): è possibile prelevare due diverse tensioni da 3,35 V e da 5 V;
- Il pin Vin: permette di realizzare sistemi alternativa per alimentare Arduino;
- **Pin analogici** (6): sono sei e sono numerati da 0 a 5 preceduti dalla lettera dell'alfabeto A. Ad essi è possibile collegare componenti come

potenziometri, sensori analogici e così via. Questi pin possono leggere il segnale da un sensore analogico (come un sensore di temperatura) e convertirlo in un valore digitale.

- Il pin AREF (9): è utilizzato per impostare una tensione di riferimento esterna (tra 0 e 5 Volt) come il limite superiore per i pin di ingresso analogico.
- **Pin digitali (7,8)**: possono essere configurati come Input o Output, sono 14 e sono numerati da 0 a 13. A questi pin possono essere collegati vari sensori, tastierini numerici e così via. I pin digitali 3, 5, 6, 9, 10 e 11 sono pin PWM (pulse with modulation).

2.2.2 BUS CAN shield

Gli shield Arduino [7] sono schede di espansione che si inseriscono nella scheda Arduino per semplificare il processo di integrazione.
Consentono di potenziare i progetti basati su Arduino, aggiungendo nuove funzionalità e capacità. La BUS CAN shield è stata progettata per la comunicazione e il controllo di dispositivi elettronici tramite il BUS CAN.

Sono disponibili diversi tipi di shield compatibili con la scheda Arduino, ciascuno progettato per soddisfare specifiche esigenze di progetto.
In questo lavoro di tesi sono state utilizzate tre BUS CAN shield di due produttori diversi, una Sparkfun BUS CAN shield e due BUS CAN shield by Seeed Studio, in particolare una V1 e una V2.

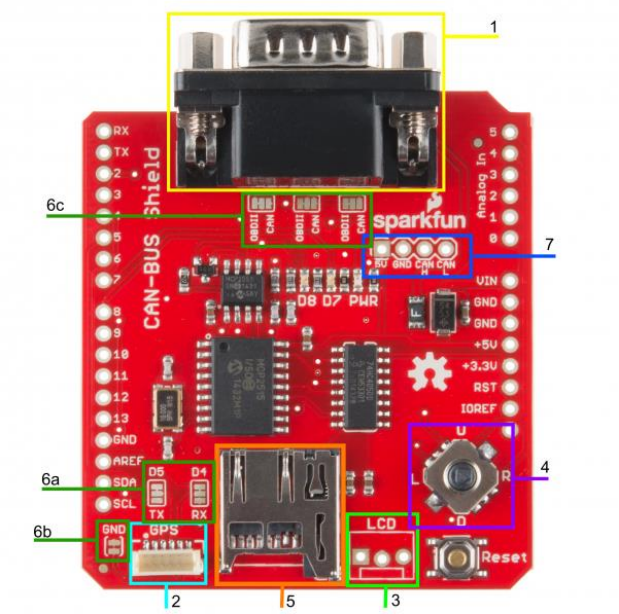


Figura 9. Sparkfun BUS CAN shield

Per la Sparkfun BUS CAN shield, in riferimento alla figura 9, si distinguono le seguenti componenti hardware [8]:

1. **DB9 Connector**: è il connettore principale della shield.
   Consente di interfacciarsi alle porte OBD-II con un cavo da DB9 a OBD-II.
2. **GPS Connector**: è un connettore integrato a 6 pin. La scheda è progettata per interfacciarsi con il ricevitore GPS EM-506 o con il ricevitore GPS GP-735. Il GND jumper consente all'utente di modificare il connettore GPS per le unità che non dispongono di una connessione sul pin 5 del connettore.
3. **LCD Connector**: L'ingombro LCD nella shield è compatibile con un male-3pin JST connector. La connessione è progettata per LCD da 5 V. L'ordine dei pin è 5V, GND e RX/D6.
4. **Joystick**: fornisce un'interfaccia utente di base per il controllo delle visualizzazioni dello schermo e la selezione delle impostazioni di scansione CAN.

Offre 5 opzioni utente di base:

- Up
- Down
- Left
- Right
- Click selection

5. **microSD Slot**: offre all'utente la possibilità di memorizzare i dati raccolti su una scheda microSD. I dati raccolti possono includere l'input dell'utente sul joystick, le informazioni BUS CAN raccolte, le uscite LCD o i dati I/O generali.
6. **Jumpers**: nella BUS CAN shield ne sono presenti sei:
   - 6a. **SJ1** e **SJ2**: consentono all'utente di selezionare tra UART e Software Seriale per far comunicare l'unità GPS con Arduino.
   - 6b. **SJ3**: consente all'utente di separare il pin 5 del connettore GPS dalla GND line; solitamente viene chiuso.
   - 6c. **SJ4**, **SJ5** e **SJ6**: consentono all'utente di selezionare la configurazione pin DB9 tra OBD-II e CAN.
7. **CAN Pins**: consentono la connessione diretta dei dati CAN provenienti dal connettore DB9. Sono:
   - 5V
   - GND
   - CAN H (CAN HIGH)
   - CAN L (CAN LOW).

Per quanto riguarda la BUS CAN shield V2, essa [9] presenta le seguenti caratteristiche:

- Circuiti integrati: controller CAN-BUS MCP2515 stand-alone e rice-trasmettitore CAN MCP2551 ad alta velocità;
- Supporta gli standard industriali: fornisce un connettore sub-D a 9 pin;
- Media velocità di comunicazione: implementa CAN V2.0B fino a 1 Mb/s;
- Lunga distanza di trasmissione: Dati standard (11 bit) ed estesi (29 bit) e frame remoti;
- Due buffer di ricezione con archiviazione dei messaggi con priorità.

Le componenti hardware della BUS CAN shield by Seeed Studio sono mostrate in figura 10.

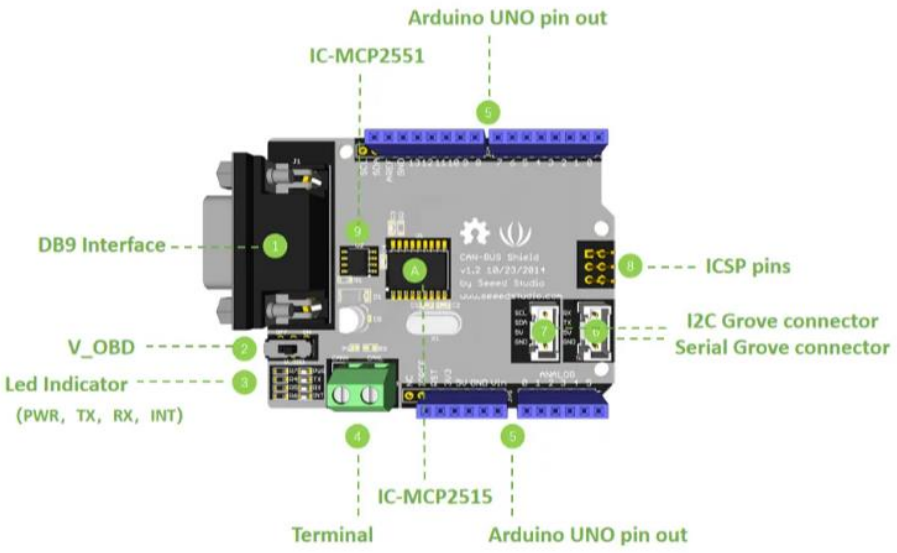


Figura 10. Hardware del BUS CAN shield by Seeed Studio

Capitolo 3: Software e programmazione

Riguardo alla parte software [4] di cui si compone la scheda Arduino si parla di IDE (Integrated Development Environment). Il linguaggio di programmazione utilizzato per la scheda è una variante del linguaggio C/C++. La programmazione della scheda è semplice e permette di essere utilizzata anche da chi non conosce propriamente il linguaggio di programmazione.

Una delle caratteristiche più utili dell'editor è la sua capacità di eseguire una verifica automatica della correttezza del programma, identificando eventuali errori di sintassi o problemi logici direttamente durante la fase di scrittura.

L'IDE fornisce un ambiente integrato che semplifica notevolmente il lavoro di programmazione per il feedback istantaneo che fornisce al codice inserito, permettendo una maggiore produttività e una più rapida risoluzione dei problemi.

I programmi sul software in Arduino vengono chiamati *sketch*, come mostrato in figura 11.

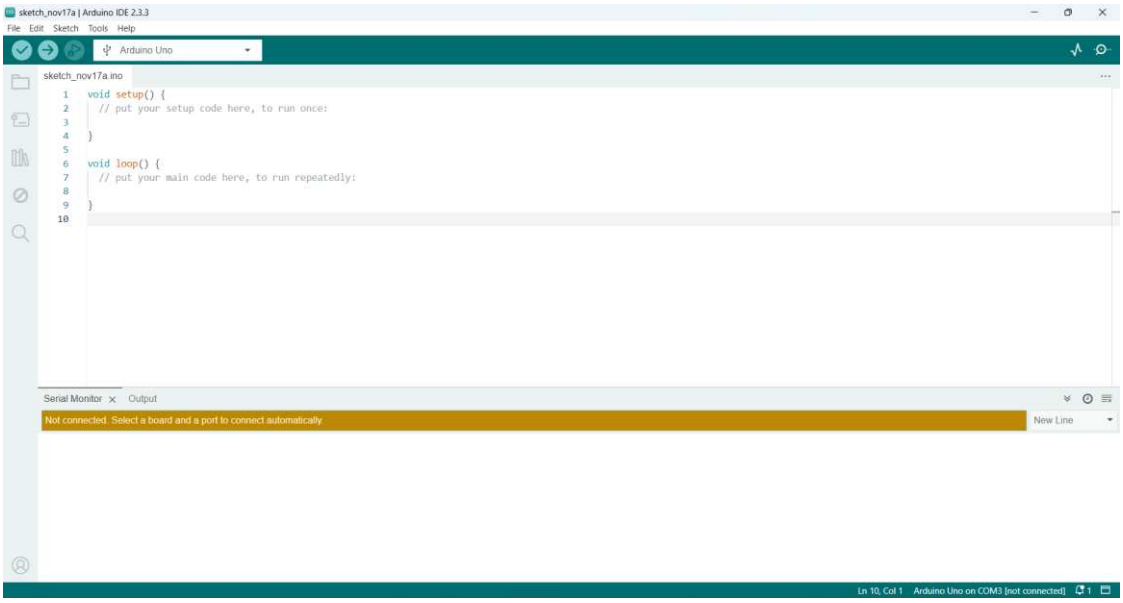


Figura 11. Esempio di uno sketch di Arduino

Il codice viene definito attraverso 2 funzioni:

- Void setup(): rappresenta la sezione del codice che si attiva una sola volta e non viene ripetuta fino al riavvio della scheda o al caricamento di un nuovo programma.
- Void loop(): indica la parte del codice che viene eseguita in modo continuo, fino a che la scheda non viene riprogrammata o spenta.

Il corpo di una funzione è definito mediante le parentesi graffe.
Dopo aver completato la scrittura del codice, il passo successivo consiste nel fare clic sul pulsante di *upload* per trasferire il programma sull'Arduino.
Una volta che il caricamento è stato eseguito correttamente, è possibile osservare il risultato dell'esecuzione del codice direttamente nella sezione del *Serial monitor*.

Per facilitare la scrittura del codice [10], si ricorre frequentemente all'uso di librerie, ossia sezioni di codice che consentono di eseguire operazioni come interfacciamenti con LCD, sensori o altri moduli in modo più agevole.
Arduino contiene al suo interno già alcune librerie, ma è possibile aggiungerne altre.

Il progetto ha comportato un'analisi dettagliata delle potenzialità della scheda Arduino, ponendo particolare attenzione all'impiego di varie librerie e all'utilizzo di codici per le applicazioni previste.

3.1 Configurazione Arduino R3 UNO

Dopo aver connesso il cavo USB collegato all'Arduino R3 UNO al computer, per la configurazione della scheda è necessario installare Arduino AVR Boards dalla *boards managers*, come indicato nella figura 12, e selezionare la porta Arduino Uno, come indicato nella figura 13.

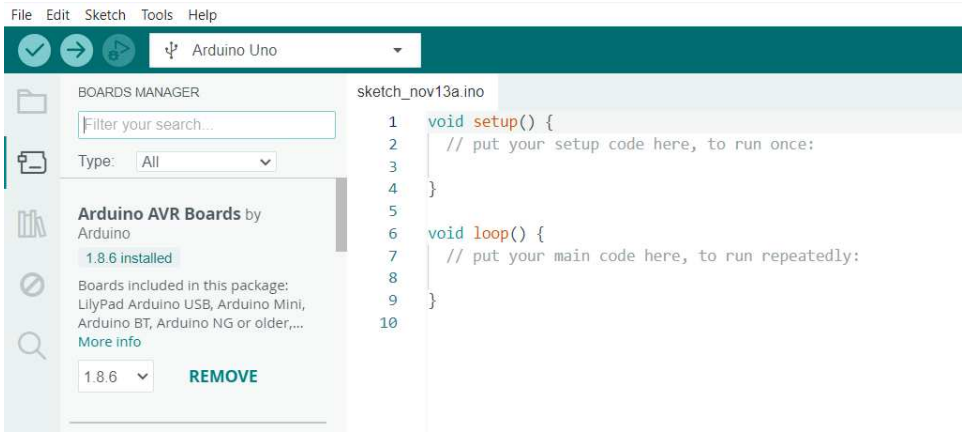


Figura 12. Installazione Arduino AVR Boards

Figura 13. Selezione della porta Arduino Uno

3.2 Lettura della tensione tramite jumper

I jumpers sono cavi di collegamento usate nelle schede elettroniche per stabilire connessioni elettriche tra vari pin o circuiti.

Connettendo un'estremità del jumper a uno dei pin analogici (nel seguente lavoro è stato utilizzato il PIN A0) e l'altra estremità nel pin 5V (entrambi i pin presenti nell'Arduino R3 UNO), è stato possibile leggere il valore della tensione di 5 V attraverso il codice mostrato in figura 14.

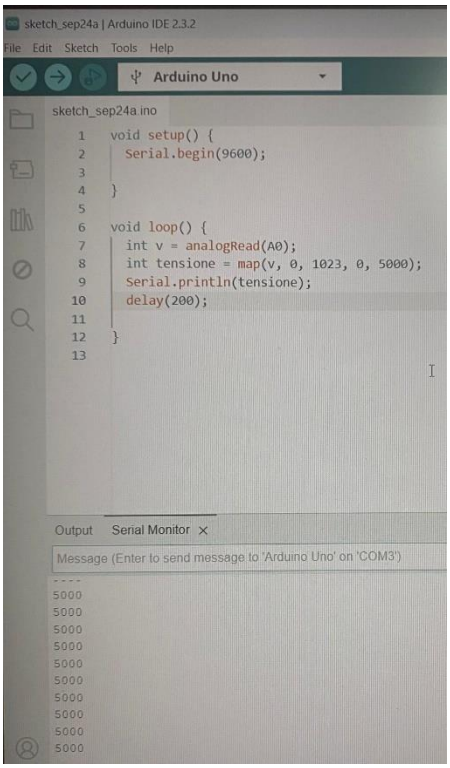


Figura 14. Codice di lettura della tensione

Nella parte del void setup() il codice indica la velocità di trasmissione dei dati pari a 9600 bps, consentendo al microcontrollore di ricevere dati tramite la porta seriale.
Nella parte del void loop() il codice invece permette la lettura del valore di tensione e la stampa sul monitor dei valori captati.
In particolare:

- int v = analogRead(A0): consente di leggere il valore analogico del pin A0;
- int tensione = map(v, 0, 1023, 0, 5000): serve a convertire il valore letto (compreso tra 0 e 1023) in un valore compreso tra 0 e 5000, ossia un valore di 0 corrisponderà a 0 mV e un valore di 1023 corrisponderà a 5000 mV (5V) dato che la scheda utilizzata funziona a 5V;
- Serial.print(tensione): stampa il valore della variabile tensione sulla porta seriale;
- delay(200): rende il programma in pausa per 200 millisecondi (0,2 secondi) prima di riprendere l'esecuzione dall'inizio del ciclo loop. Questo impedisce che il monitor seriale sia sovraccaricato da un eccesso di dati in un breve intervallo di tempo.

Successivamente, sono state salvate le informazioni presenti nel serial monitor mediate Coolterm, che è un'applicazione usata per la comunicazione seriale con dispositivi elettronici, come ad esempio microcontrollori, modem e altri dispositivi che utilizzano la comunicazione via porta seriale [11], come mostrato in figura 15.

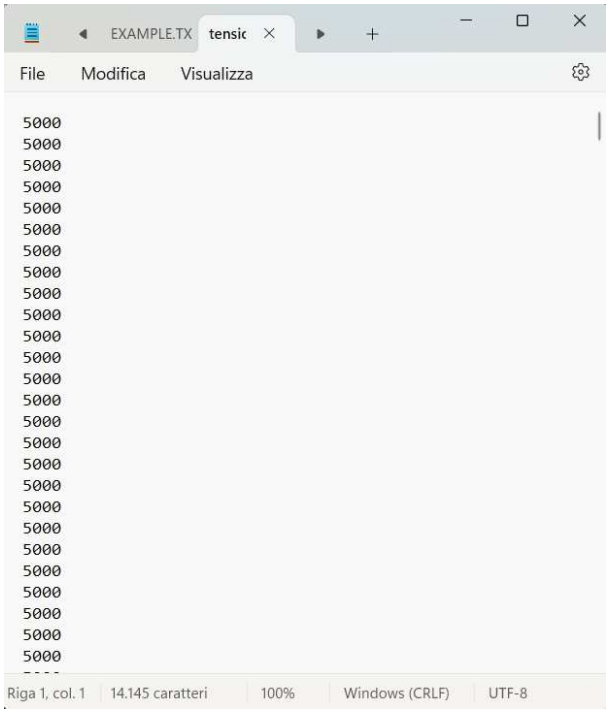


Figura 15. Valori della tensione nel software Coolterm

3.3 Configurazione BUS CAN shield

Una volta che la Sparkfun BUS CAN shield è stata inserito sopra l'Arduino R3 UNO e connessa tramite i pin, bisogna eseguire la configurazione della shield.

È necessario scaricare la libreria specifica progettata per la scheda BUS CAN shield, mostrata in figura 16, e aggiungerla nelle librerie già esistenti su Arduino tramite la funzionalità sketch, come mostrato in figura 17.

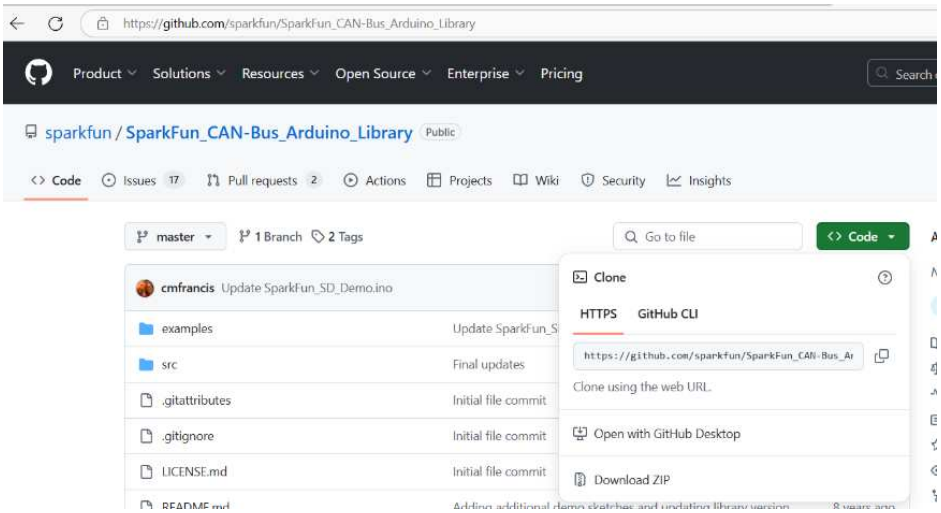


Figura 16. Libreria utile alla BUS CAN shield

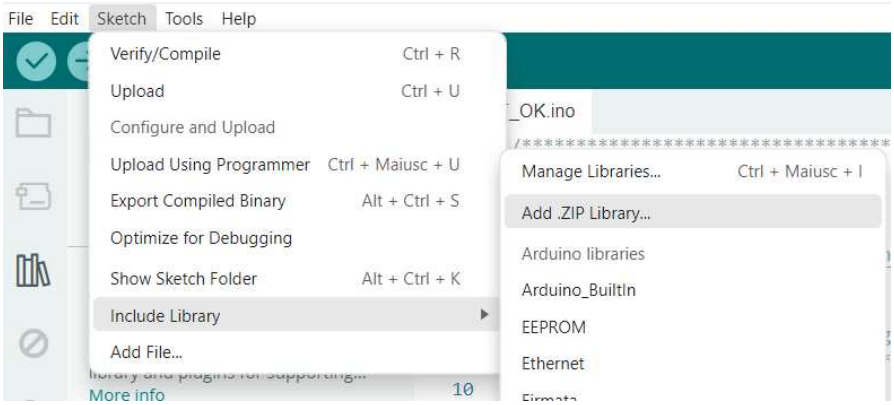


Figura 17. Funzione sketch per aggiungere la libreria scaricata

Attraverso il codice indicato in figura 18, è stato possibile eseguire la configurazione del BUS CAN SHIELD.

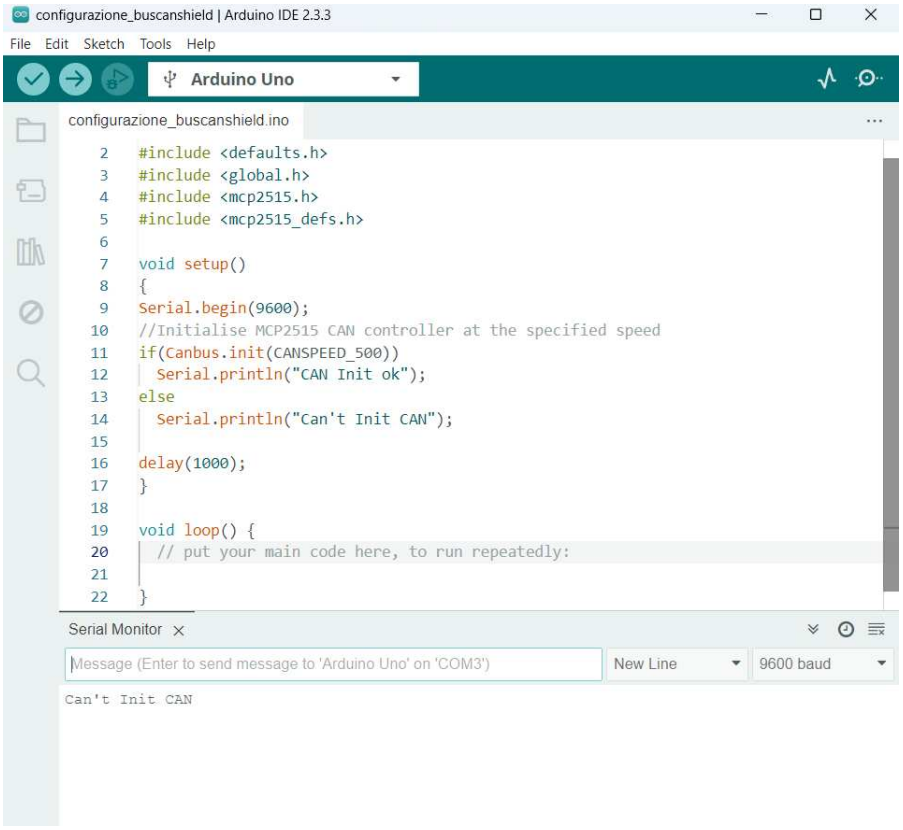


Figura 18. Codice necessario alla configurazione dello shield

Capitolo 4: Acquisizione dati

Il capitolo seguente sarà dedicato alla descrizione dell'acquisizione dei dati.

4.1 Lettura delle posizioni del joystick sulla BUS CAN shield e salvataggio sulla scheda microSD

Attraverso l'utilizzo di Arduino, è stato possibile implementare un codice che consenta la lettura delle posizioni del joystick integrato nella Sparkfun BUS CAN shield. Inoltre, è stata inserita una scheda microSD nella shield, permettendo di salvare i risultati ottenuti dal codice tramite un file CoolTerm per la visualizzazione dei dati.
Tale codice, riportato in seguito, non solo ha permesso la lettura delle posizioni del joystick, ma ha anche integrato la funzionalità di salvataggio su scheda microSD.

```
1   #include <SPI.h>
2   #include <SD.h>
3
4   const int chipSelect = 9;
5
6   #define joy_x A0
7   #define joy_y A1
8
9   int x= 0;
10  int y= 0;
11
```

Nella parte della sezione di definizione sono state incluse le librerie SPI e SD per comunicare con la scheda SD utilizzando il protocollo SPI. Inoltre, il chipSelect è stato impostato su 9, che rappresenta il pin utilizzato per

selezione la scheda SD. Sono stati definiti joy_x e joy_y come i pin analogici A0 e A1, i quali sono utilizzati per leggere i valori degli assi X e Y di un joystick.

```
12  void setup()
13  {
14    // Open serial communications and wait for port to open:
15    Serial.begin(9600);
16    while (!Serial) {
17      ; // wait for serial port to connect. Needed for Leonardo only
18    }
19
20    Serial.print("Initializing SD card...");
21    // make sure that the default chip select pin is set to
22    // output, even if you don't use it:
23    pinMode(chipSelect, OUTPUT);
24
25    // see if the card is present and can be initialized:
26    if (!SD.begin(chipSelect)) {
27      Serial.println("Card failed, or not present");
28      // don't do anything more:
29      return;
30    }
31    Serial.println("card initialized.");
32
33    pinMode (joy_x, INPUT);
34    pinMode (joy_y, INPUT);
35  }
36
```

Nella parte del void setup() il codice indica la velocità di trasmissione dei dati pari a 9600 bps. Una volta che la porta seriale si connette, si configura la scheda SD. Se la scheda SD non riesce a essere configurata, viene stampato un messaggio di errore. Se invece la configurazione della scheda SD è processata correttamente, viene stampato dal programma “card initialized”. Sono inoltre stati imposti i pin del joystick come ingressi.

```
37  void loop()
38  {
39    String dataString = "";
40    File dataFile = SD.open("datalog.txt", FILE_WRITE);
41    if (dataFile)   {
42      int timeStamp = millis();
43      dataFile.print(timeStamp);
44      dataFile.print(" ms");
45      dataFile.print(", ");
46      Serial.print(timeStamp);
47      Serial.print(",");
48
49      // Joystick asse x
50      Serial.print("  |||  ");
51      Serial.print("X: ");
52      x = analogRead(joy_x);
53      Serial.println(x);
54      if ( x >= 0)
55      // Joystick asse y
56      Serial.print("  |||  ");
57      Serial.print("Y: ");
58      x = analogRead(joy_y);
59      Serial.println(y);
60      if ( y >= 0)
61
62      dataFile.println(); //create a new row to read data more clearly
63      dataFile.close();   //close file
64      Serial.println();   //print to the serial port too:
65    }
66    // if the file isn't open, pop up an error:
67    else
68    {
69      Serial.println("error opening datalog.txt");
70    }
71  }
```

Nella parte del void loop() il codice inizia aprendo il file datalog.txt oppure crearlo se il file non esiste. Se il file è aperto correttamente, legge il valore corrente di millis(), che rappresenta il tempo trascorso dall'accensione del programma, e lo scrive nel file e nella porta seriale. Poi la funzione analogRead() consente la lettura e la stampa dei valori delle posizioni sul joystick sull'asse x e y. Ogni volta che vengono letti i valori, viene creato una nuova riga nel file per garantire che i dati siano formattati correttamente. È fondamentare poi chiudere il file tramite la funzione dataFile.close().

Nel caso di errore nell'apertura del file datalog.txt, viene stampato "error opening datalog.txt".

4.2 Lettura dati dal simulatore ECU automobilistico tramite cavo DB9

Il lavoro di tesi ha previsto l'utilizzo di un simulatore a pacchetti ECU (Electronic Control Unit) progettato per emulare il comportamento delle diverse unità di controllo elettronico utilizzate nei veicoli, mostrato in figura 19.

Alcune delle principali ECU presenti nel simulatore sono:

- ECU del motore (Engine RPM): gestisce l'iniezione di carburante, l'accensione, il controllo delle emissioni e altre funzioni del motore;
- ECU trasmissione (Vehicle Speed Sensor): regola il cambio delle marce, controllando i parametri della trasmissione automatica;
- ECU di controllo della stabilità (Ignition Advance Angle): gestisce la stabilità del veicolo, prevenendo il sovrasterzo e il sottosterzo;
- ECU di alimentazione (Air/Fuel Ratio): regola la formazione della miscela aria-carburante.

Figura 19. Simulatore a pacchetti utilizzato

Sono stati impiegati Arduino R3 Uno collegato alla Sparkfun BUS CAN shield e un cavo DB9 per stabilire la connessione con il simulatore.

In primo luogo, è stato utilizzato un esempio di codice presente nella libreria SparkFun CAN-Bus precedentemente scaricata, come mostrato in figura 20.

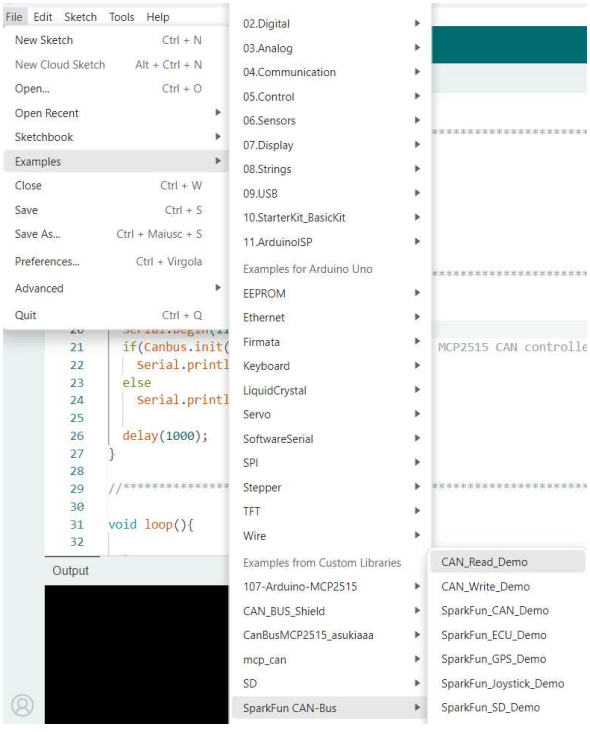


Figura 20. Esempio utilizzato per la lettura dati del simulatore

Tuttavia, durante la fase di connessione e lettura dei dati dal simulatore Arduino sembrava riconoscere solo il BUS CAN shield e nessun dato proveniente dal simulatore è stato stampato nel serial monitor del codice utilizzato.
Il fallimento nella lettura dei dati dal simulatore potrebbe essere attribuito a una combinazione di errori di configurazione, problemi di cablaggio, malfunzionamenti del simulatore o incompatibilità tra i vari dispositivi impiegati nel sistema.

4.3 Connessione tra due BUS CAN shield

È stata realizzata una connessione tra due BUS CAN shield by Seeed Studio utilizzando due jumpers, come riportato in figura 21.

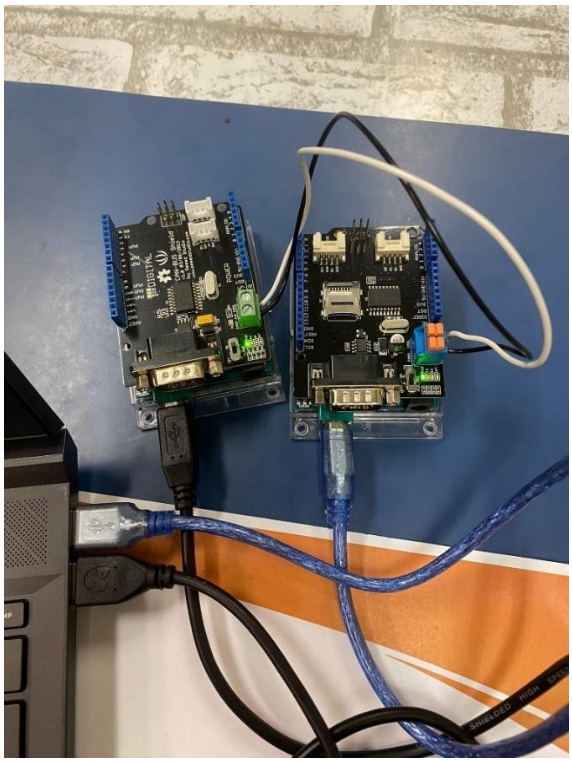


Figura 21. Connessione tra due BUS CAN shield

In questo contesto si distingue il dispositivo che funge da *master* e quello che opera come *slave*. Master-slave è [12] un modello di comunicazione per dispositivi hardware in cui un dispositivo ha un controllo unidirezionale su uno o più dispositivi.

Il *master* opera come istanza di controllo in maniera autonoma e assegna allo *slave* i permessi necessari, mentre lo *slave* agisce in maniera passiva su richiesta del *master*. Questo tipo di comunicazione impedisce lo scambio di informazioni non autorizzato e permette di controllare le risorse gestite.

Per avviare la connessione, è stata innanzitutto scaricata una libreria dedicata che supporta le operazioni necessarie per stabilire e gestire la comunicazione tra i due BUS CAN shield, mostrata in figura 22. Questa libreria contiene le funzioni e le strutture dati necessarie per inviare e ricevere messaggi in modo conforme agli standard del protocollo CAN.

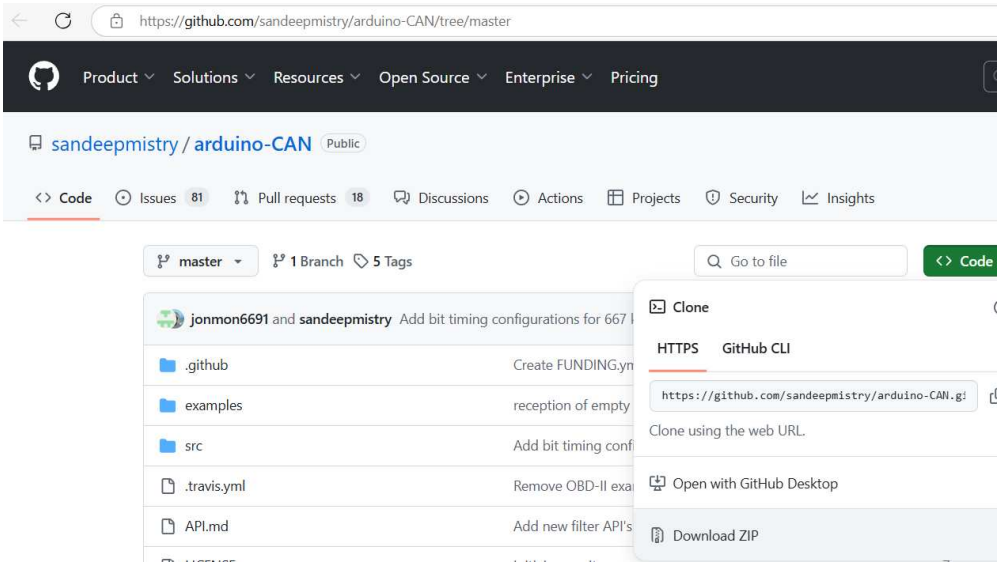


Figura 22. Libreria necessaria alla connessione tra due BUS CAN shield

GitHub - sandeepmistry/arduino-CAN: An Arduino library for sending and receiving data using CAN bus.

Successivamente, sono stati utilizzati due esempi di codice presenti nella libreria installata su Arduino, uno per il *master* e uno per lo *slave*, mostrati in figura 23.

Nel codice del *master* sono stati inviati pacchetti CAN utilizzando un ID standard e un ID esteso, mentre nel codice dello *slave* le informazioni ricevute dal *master*, che possono essere considerate come comandi o dati di controllo, sono state elaborate e codificate utilizzando il codice ASCII (American Standard Code for Information Interchange).

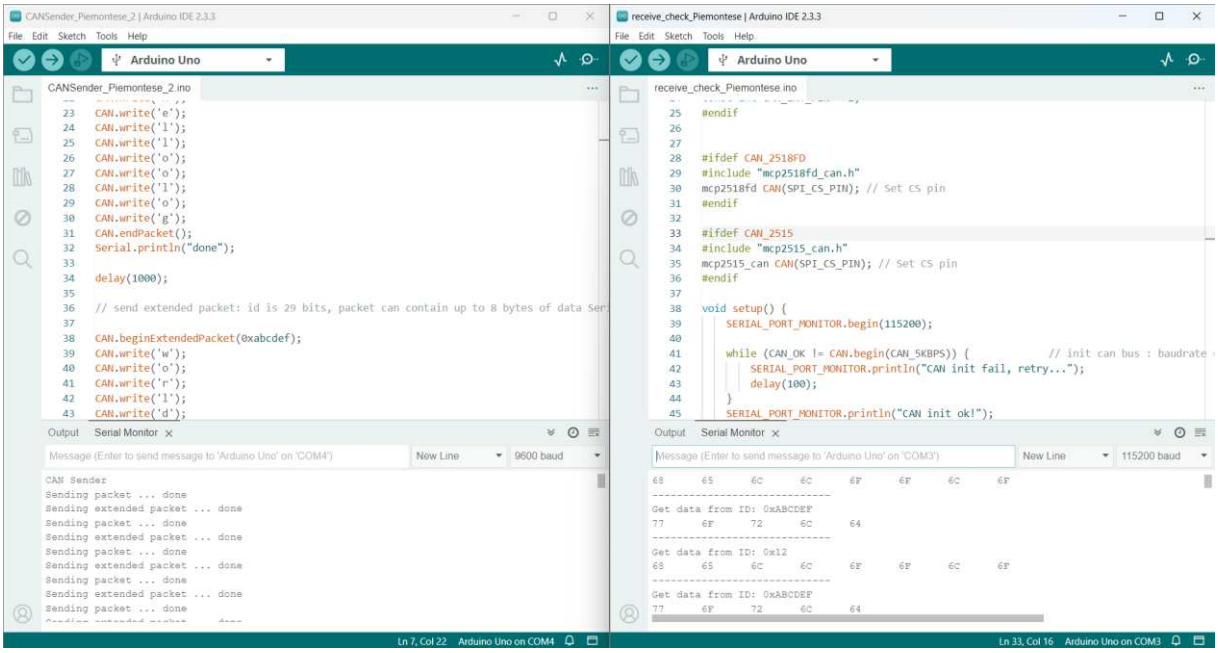
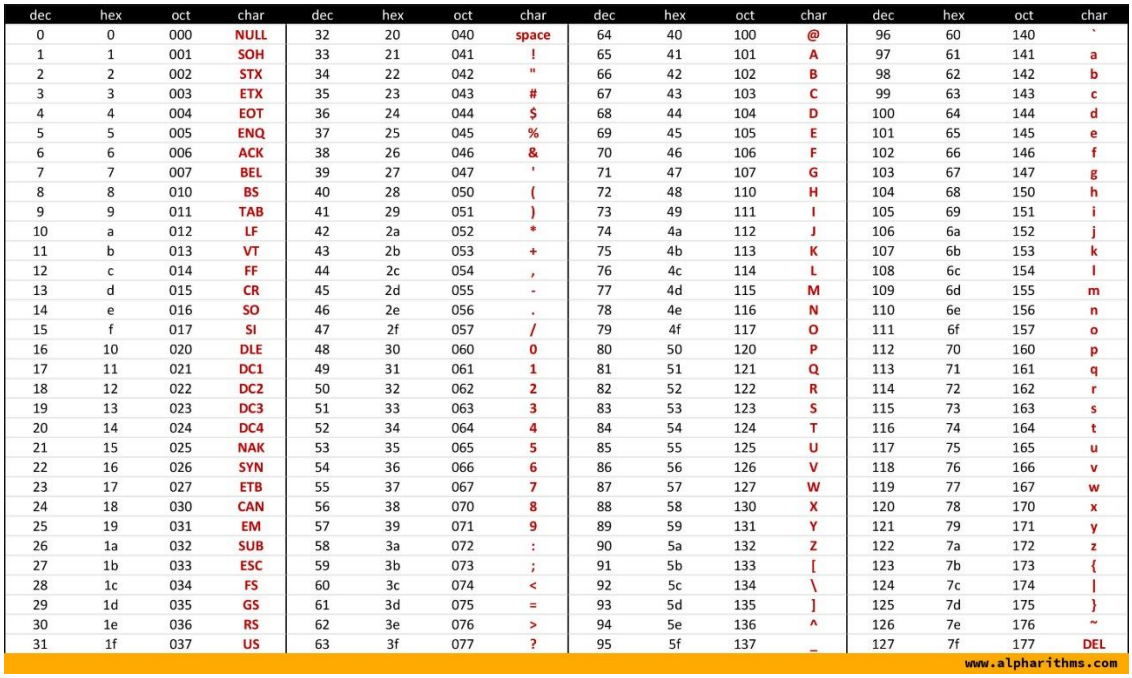


Figura 23. Esempi utilizzati per la connessione tra due BUS CAN shield

Il codice ASCII [13] è uno standard che assegna un numero univoco a ogni carattere testuale (come mostrato in figura 24) utilizzato nei computer e, più in generale, dai sistemi e dispositivi di informatica e telecomunicazione.



dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char
0	0	000	NULL	32	20	040	space	64	40	100	@	96	60	140	`
1	1	001	SOH	33	21	041	!	65	41	101	A	97	61	141	a
2	2	002	STX	34	22	042	"	66	42	102	B	98	62	142	b
3	3	003	ETX	35	23	043	#	67	43	103	C	99	63	143	c
4	4	004	EOT	36	24	044	$	68	44	104	D	100	64	144	d
5	5	005	ENQ	37	25	045	%	69	45	105	E	101	65	145	e
6	6	006	ACK	38	26	046	&	70	46	106	F	102	66	146	f
7	7	007	BEL	39	27	047	'	71	47	107	G	103	67	147	g
8	8	010	BS	40	28	050	(	72	48	110	H	104	68	150	h
9	9	011	TAB	41	29	051	)	73	49	111	I	105	69	151	i
10	a	012	LF	42	2a	052	*	74	4a	112	J	106	6a	152	j
11	b	013	VT	43	2b	053	+	75	4b	113	K	107	6b	153	k
12	c	014	FF	44	2c	054	,	76	4c	114	L	108	6c	154	l
13	d	015	CR	45	2d	055	-	77	4d	115	M	109	6d	155	m
14	e	016	SO	46	2e	056	.	78	4e	116	N	110	6e	156	n
15	f	017	SI	47	2f	057	/	79	4f	117	O	111	6f	157	o
16	10	020	DLE	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM	57	39	071	9	89	59	131	Y	121	79	171	y
26	1a	032	SUB	58	3a	072	:	90	5a	132	Z	122	7a	172	z
27	1b	033	ESC	59	3b	073	;	91	5b	133	[	123	7b	173	{
28	1c	034	FS	60	3c	074	<	92	5c	134	\	124	7c	174	|
29	1d	035	GS	61	3d	075	=	93	5d	135	]	125	7d	175	}
30	1e	036	RS	62	3e	076	>	94	5e	136	^	126	7e	176	~
31	1f	037	US	63	3f	077	?	95	5f	137	_	127	7f	177	DEL

www.alpharithms.com

Figura 24. Tabella codice ASCII

Nello sketch utilizzato dal *master* vengono stampati due messaggi.
Il codice scrive una serie di caratteri ('h', 'e', 'l', 'l', 'o', 'o', 'l', 'o', 'g') nel primo pacchetto, che ha un ID di 11 bit specificato da 0x12 (18 in decimale).
Il primo pacchetto del codice può contenere fino a 8 byte di dati, quindi in questo caso si stanno trasmettendo 9 caratteri, ma solo i primi 8 sono inviati.
Il secondo pacchetto presente nel codice del master è il pacchetto esteso, che presenta un ID di 29 bit, specificato come 0xabcdef. È presente il contenuto ('w', 'o', 'r', 'l', 'd') nella memoria del pacchetto esteso.

I codici utilizzati hanno svolto un ruolo fondamentale nell'analisi delle funzionalità e dei limiti dei pacchetti CAN, permettendo di esplorare in modo approfondito le capacità di comunicazione tra i dispositivi.
Grazie a tali codici, è stata implementata la connessione tra due BUS CAN shield, i quali hanno interagito in modo efficace per scambiare informazioni.

È stato possibile visualizzare in tempo reale il messaggio ricevuto dallo *slave*. Questo ha reso possibile non solo monitorare le comunicazioni in corso, ma anche comprendere come il messaggio inviato dal *master*, il dispositivo che funge da centrale nel sistema, venisse codificato e interpretato dallo *slave*.

Conclusioni

In conclusione, il progetto di tesi ha constatato l'efficacia di un sistema a microcontrollore dedicato all'acquisizione di parametri diagnostici per veicoli, sfruttando la rete di comunicazione CAN (Controller Area Network). Questo approccio è particolarmente vantaggioso nel contesto della diagnostica automobilistica, dove la capacità di monitorare e analizzare in tempo reale i dati provenienti dai veicoli è diventata sempre più cruciale.

Nella presente tesi, si è esposta, in primo luogo, una parte sulle nozioni riguardanti il protocollo CAN e la scheda Arduino.
In secondo luogo, sono stati illustrati i passaggi della fase di realizzazione del seguente progetto di tesi. L'attività pratica di tirocinio è partita dalla configurazione di Arduino R3 UNO per poi proseguire con l'acquisizione dei dati tramite l'implementazione di codici su Arduino. Oltre alla lettura dati, attraverso la connessione tra due BUS CAN shield è stato sperimentato l'invio di dati tramite il protocollo CAN.
Il progetto ha dimostrato come sia possibile non solo acquisire e archiviare informazioni in tempo reale, ma anche gestire in modo efficace la comunicazione tra dispositivi differenti all'interno della rete CAN.

A partire da questo progetto di tesi, si può delineare uno sviluppo futuro che si concentri sull'analisi approfondita delle problematiche che hanno ostacolato la lettura dei dati provenienti da un simulatore a pacchetti. Questa comprensione è fondamentale poiché offre la possibilità di identificare le cause specifiche che hanno portato a tali difficoltà.

Un obiettivo primario sarà quello di analizzare in dettaglio i vari aspetti tecnici e operativi coinvolti nel processo di acquisizione e monitoraggio dei dati.

Attraverso questa analisi si potrebbero sviluppare soluzioni mirate per ottimizzare il trasferimento dei dati e garantire una lettura più precisa e affidabile.

Bibliografia

[1] What Is CAN Bus (Controller Area Network) and How It Compares to Other Vehicle Bus Network
What Is Can Bus (Controller Area Network) | Dewesoft

[2] CAN BUS Controller Area Network
Microsoft PowerPoint - SCA04_CAN Bus 13-14.ppt [modalità compatibilità]

[3] BOSCH'S CONTROLLER AREA NETWORK
CAN BUS in applicazioni Automobilistiche

[4] Che cos'è Arduino? Come funziona?
cosa è arduino? come funziona arduino? - PROGETTI ARDUINO

[5] Introduzione ad Arduino: una guida per principianti
Introduzione ad Arduino: una guida per principianti - KnowHow

[6] Arduino: cos'è, come funziona e per quali scopi si usa
Arduino: cos'è, come funziona e per quali scopi si usa - Moreware Blog

[7] Tutto ciò che dovete sapere sulle schermature Arduino
Tutto ciò che dovete sapere sulle schermature Arduino - KnowHow

[8] CAN-BUS Shield Hookup Guide
CAN-BUS Shield Hookup Guide - SparkFun Learn

[9] CAN-BUS Shield V2.0
CAN-BUS Shield V2.0 | Seeed Studio Wiki

[10] GUIDA ARDUINO 6: LE LIBRERIE ARDUINO
GUIDA ARDUINO 6: LE LIBRERIE ARDUINO | NE555

[11] Roger Meier's Freeware
Roger Meier's Freeware

[12] Che cosa si nasconde dietro l'architettura master-slave?
Master-slave: che cos'è l'architettura master-slave? - IONOS

[13] Carattere codice ASCII: tabella completa
Caratteri codice ASCII: tabella completa • Scuolissima.com