FIRENZE RACE TEAM

FR-24
COMBUSTION &
DRIVERLESS

Sviluppo e Analisi Dettagliata del Software per la Main Control Unit nelle Applicazioni di Formula SAE

Corso di Software Engineering for Embedded Systems

Candidati
Lorenzo Porcheddu
Emanuele Nencioni

Docente
Prof.ssa Laura Carnevali

UNIVERSITÀ
DEGLI STUDI
FIRENZE

DINFO
DIPARTIMENTO DI
INGEGNERIA DELL'INFORMAZIONE

Sommario

Indice delle Figure

Introduzione

Il Firenze Race Team è la squadra ufficiale di Formula SAE e Formula Student dell'Università degli Studi di Firenze. È un laboratorio didattico aperto agli studenti di tutte le facoltà dell'ateneo con lo scopo di progettare e realizzare una monoposto da competizione.

Visti i prossimi cambiamenti di regolamento delle competizioni di Formula Student e il futuro del mercato automobilistico, il Team ha iniziato a sviluppare il progetto di **auto a guida autonoma**.

L'obiettivo principale è quello di partecipare nel 2024 alla competizione di FS-East con la vettura a combustione, per poi scendere in pista nella categoria 1D di Formula ATA con la vettura *driverless*.

Canditati

Emanuele Nencioni: Head of Autonomous Department – Vision & System Integration
Lorenzo Porcheddu: Head of Electronics & Controls Department

Presentazione del Progetto e Obiettivi

Nel nostro progetto per la Formula Student, stiamo lavorando a un software destinato a una scheda elettronica per auto da competizione. Quest'auto ingloba elementi tradizionali con la guida autonoma. Il nostro obiettivo è semplice: sviluppare un sistema di controllo che gestisca motore, velocità, sterzo, freni e non solo, per migliorare prestazioni e sicurezza.

La sfida più grande è fare in modo che l'auto possa guidare da sola, seguendo regole precise: questo implica l'uso di algoritmi di intelligenza artificiale e sensori avanzati per comprendere l'ambiente circostante e prendere decisioni di guida in maniera autonoma. La nostra scheda elettronica deve fare da tramite tra il computer del sistema autonomo e i suoi componenti, come l’alimentazione e i sensori, interpretando i dati raccolti e rispondendo prontamente per mantenere il tutto efficiente e sicuro.

Anche l'interazione con il pilota è fondamentale; il nostro software dovrà fornire aggiornamenti in tempo reale sullo stato dell'auto e permettere al pilota di intervenire se necessario. Ci stiamo concentrando su un'interfaccia utente che sia chiara e facile da usare.

In breve, stiamo creando un sistema che permetta a un'auto a combustione interna di guidarsi da sola in una competizione, tenendo sempre conto dell'importanza della sicurezza e delle prestazioni. È una sfida interessante che necessita di mettere insieme conoscenze di software, elettronica ed intelligenza artificiale.

Architettura del Veicolo

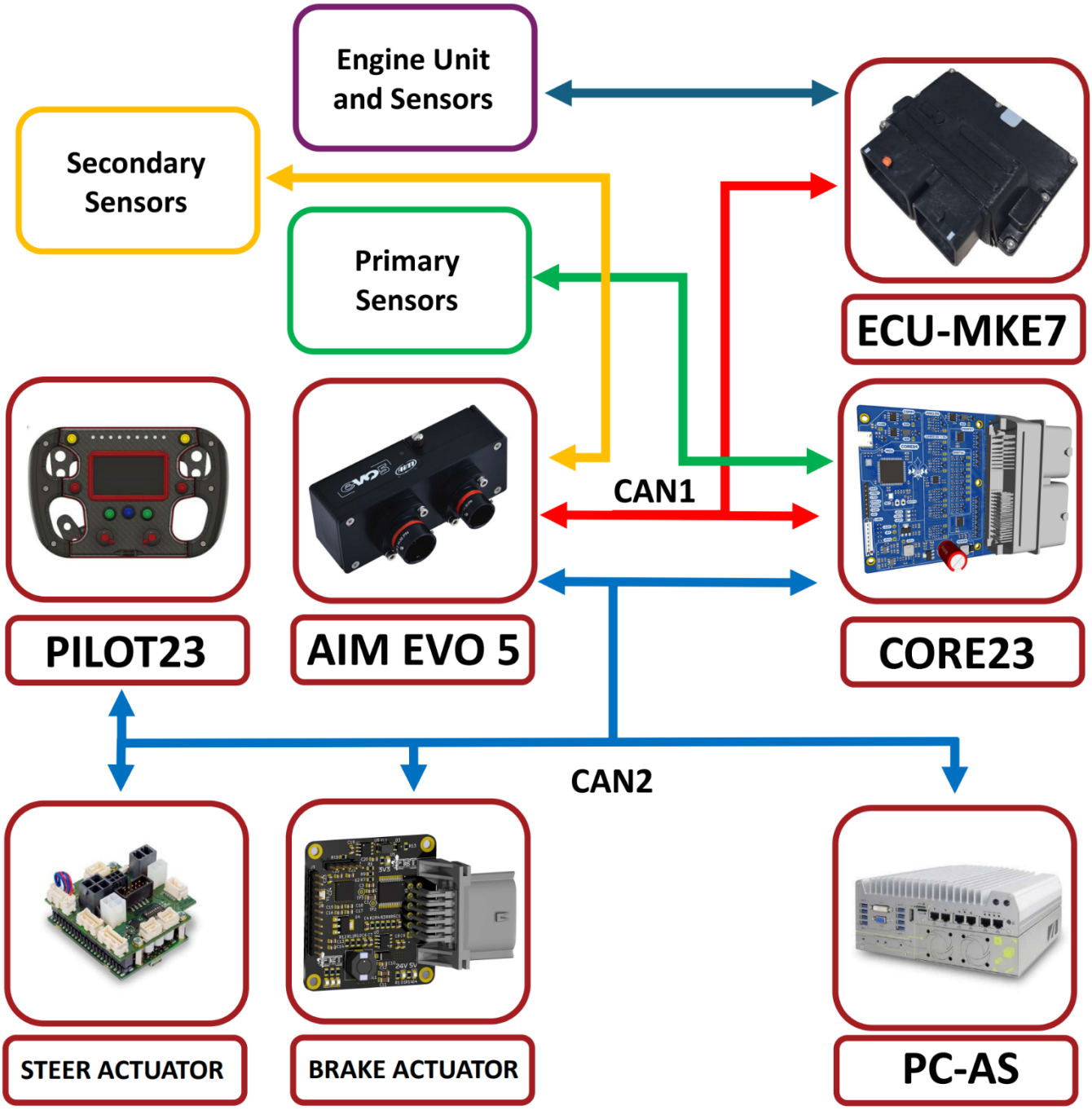

Figura 1: Architettura del Veicolo

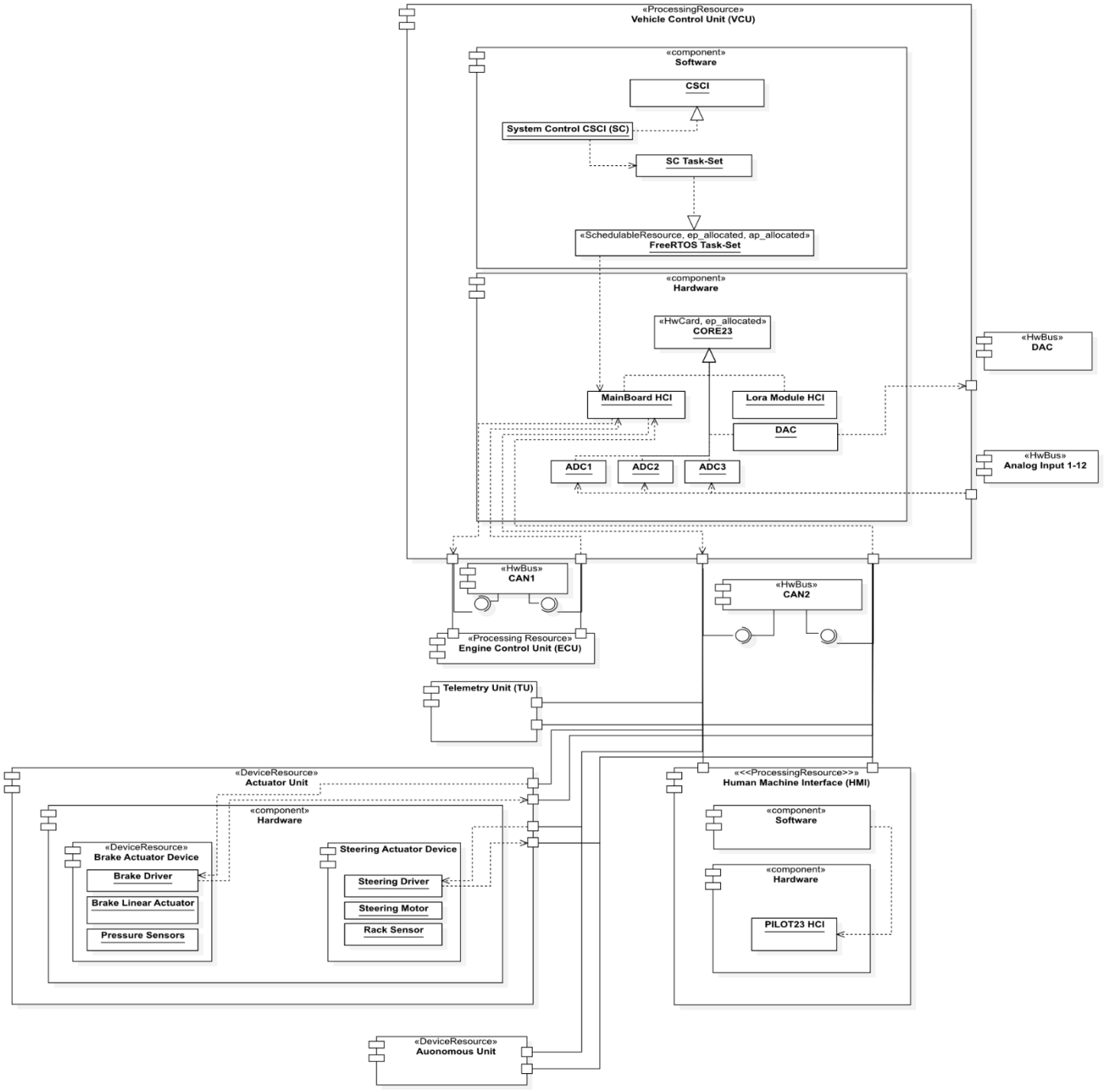

Figura 2: High Level Diagram

Come si può notare sul diagramma di alto livello, la nostra auto integra diverse centraline elettroniche; sono presenti anche due linee CAN bus per l'interconnessione, una dedicata principalmente al motore e l'altra invece al sistema autonomo. Questa suddivisione si rende necessaria data la limitata larghezza di banda disponibile, pari a 1Mbit/s di massimo teorico per ogni linea. In realtà si cerca di non superare il 50% di occupazione della banda in condizioni normali per evitare la saturazione del bus ed il conseguente rallentamento di tutti i messaggi.

Di seguito sono elencati i componenti del nostro sistema:

- **CORE23**: si tratta della MCU (Main Control Unit), ovvero della scheda principale di controllo del veicolo. Tutte le altre centraline fanno sempre riferimento a lei per gestire le funzioni interconnesse. Possiede 40 I/O tra analogiche e digitali, che permettono di collegare un'importante quantità di sensori. Avendo entrambe le linee CAN bus collegate, la scheda è in grado di ricevere una grande quantità di dati anche dai sensori non direttamente collegati. Per esempio, può leggere la temperatura del motore dalla ECU (Engine Control Unit) e inviarla alla scheda del volante, la PILOT23, per far sì che il pilota sia in grado monitorarla; viceversa, può leggere il segnale proveniente dal pedale dell'acceleratore e inviarlo alla ECU, che poi lo elaborerà secondo le varie strategie per attuare la valvola a farfalla. Su questa scheda si concentrerà lo sviluppo del software.

- **PILOT23**: si tratta della HMI (Human Machine Interface) del veicolo: infatti, posizionata all'interno del volante, si occupa principalmente di gestire l'interazione con il pilota. Sul volante è collocato un display attraverso il quale possono essere visualizzate varie informazioni come marcia inserita, velocità, giri motore, temperatura dell'acqua, tempo su giro, ecc. Inoltre, la PILOT23 si occupa di leggere i pulsanti ma soprattutto i paddles del cambio e la leva della frizione, fondamentali per interagire con il cambio.

- **MekTronic MKE7**: si tratta della ECU (Engine Control Unit). Come suggerisce il nome, è la centralina che si occupa della gestione del motore a combustione interna; ad essa sono infatti collegati tutti i sensori del motore, come temperatura acqua, pressione turbo, ecc. Possiede una linea CAN per l'interfacciamento con il resto del veicolo, attraverso la quale invia tutta la telemetria del motore in tempo reale.

- **Driver Freno e Driver Sterzo:** si tratta di schede dedicate a gestire gli attuatori del freno e dello sterzo collegate alla linea CAN per ricevere istruzioni ed inviare dati sul loro funzionamento.

- **AIM EVO 5**: si tratta di un datalogger che ha il compito di registrare e memorizzare tutte le informazioni del veicolo in tempo reale. Inoltre si occupa di memorizzare i dati di diversi sensori aggiuntivi collegati; se necessario è possibile inviare i dati di questi sensori sulle linee CAN.

- **PC autonomo:** si tratta di un PC industriale rugged in grado di eseguire le tantissime operazioni riguardanti il sistema autonomo, come ad esempio il riconoscimento dei coni stradali tramite telecamera e YOLO (You Only Look Once) oppure l'esecuzione dell'algoritmo di controllo di tipo PurePursuit. Attualmente esegue vari nodi ROS1 su Ubuntu 20.04, ma stiamo progettando un futuro update a ROS2 e Ubuntu 24.04 per migliorare prestazioni e compatibilità. Utilizza un Intel Core i7-9700 e una scheda video NVIDIA GTX1650.

Specifiche e Funzionalità

❖ **Lettura del Pedale dell'Acceleratore e Comunicazione via CAN**
- Acquisizione dei segnali analogici dal pedale dell'acceleratore.
- Conversione dei segnali analogici in dati digitali.
- Elaborazione e normalizzazione dei dati del pedale dell'acceleratore.
- Trasmissione dei dati elaborati alla centralina motore utilizzando il protocollo CAN.

❖ **Lettura delle Temperature e Gestione delle Ventole**
- Ricezione dei dati delle temperature (acqua del radiatore e aria dell'intercooler) dalla centralina motore via CAN.
- Elaborazione delle temperature ricevute per determinare i parametri operativi delle ventole.
- Controllo dell'attivazione delle ventole basato sui dati di temperatura, per ottimizzare il raffreddamento.

❖ **Telemetria tramite LoRa**
- Raccolta e elaborazione dei dati di telemetria del veicolo, inclusi posizione, velocità, e parametri di funzionamento critici.
- Codifica e trasmissione dei dati di telemetria tramite il modulo LoRa.
- Gestione della connessione e del protocollo di comunicazione LoRa per garantire l'affidabilità e la sicurezza dei dati trasmessi.

❖ **Gestione del Cambio Marce**
- Interpretazione dei comandi di cambio marcia provenienti dal conducente o dal sistema di controllo autonomo.
- Attuazione del cambio marce attraverso sistemi elettromeccanici, garantendo tempi di risposta rapidi ed innesti precisi.

❖ **Gestione della Frizione tramite Valvola Proporzionale (DAC)**
- Controllo preciso della valvola proporzionale della frizione mediante un convertitore digitale-analogico (DAC).
- Regolazione dell'apertura della valvola in base ai dati di input per una gestione ottimale dell'ingaggio della frizione.

❖ **Gestione della Comunicazione sulle Linee CAN**
- Implementazione di un gestore CAN per facilitare la comunicazione bidirezionale con i dispositivi del veicolo.
- Garanzia di interoperabilità e compatibilità con i protocolli CAN standard e personalizzati.

❖ **Sistema Autonomo e Gestione degli Stati**

- Sviluppo di un framework per il controllo autonomo del veicolo, includendo la gestione degli stati operativi e dei modi di guida.
- Implementazione di un sistema di monitoraggio e gestione degli errori per garantire la sicurezza e l'affidabilità del sistema autonomo.

❖ **Gestione del Freno di Emergenza**

- Integrazione di un sistema di frenata di emergenza attivabile automaticamente in situazioni critiche o tramite comando manuale.
- Sviluppo di algoritmi per la valutazione delle condizioni di emergenza e l'attivazione rapida e sicura dei freni.

❖ **Gestione della Brake Light**

- Integrazione di un sistema intelligente per il controllo delle luci di stop, che attiva automaticamente le luci freno in base ai dati di decelerazione rilevati dai sensori del veicolo, nonché ai comandi di frenata d'emergenza.

❖ **Gestione dei LED del Sistema Autonomo**

- Implementazione di un sistema di segnalazione a LED che segua rigorosamente le linee guida stabilite dal regolamento della Formula Student per quanto riguarda colore, modalità di lampeggio, e visibilità. Questo assicura che lo stato del sistema autonomo sia immediatamente riconoscibile da marshals, spettatori e altri partecipanti, incrementando la sicurezza in pista.
- Adattamento delle sequenze di illuminazione per indicare chiaramente le varie fasi operative del sistema autonomo, come l'avvio, l'esecuzione e l'interruzione dell'automazione, garantendo che ogni transizione sia intuitivamente comprensibile e conforme alle norme di competizione.

Task set e Activity Diagram

Usando il software StarUML abbiamo redatto un Activity Diagram e successivamente, un Object Diagram; data la dimensione dei diagrammi stessi, sono rappresentati in allegato al segeunte report.

Di seguito, i singoli task e le relative funzionalità

Pedal Task

Si tratta del task che si occupa della gestione del comando acceleratore inviato al motore a combustione; funziona solamente durante la modalità manuale, perché durante quella autonoma è attivo un altro task. Si blocca il semaforo sull'ADC per leggere il pedale dell'acceleratore, quindi vengono inviati i dati alla centralina motore attraverso il DAC. L'ultima attività che deve fare è un controllo sulla plausibilità della lettura, per motivi di sicurezza. Leggendo i due valori, se nota un discostamento superiore al 10% va in errore.

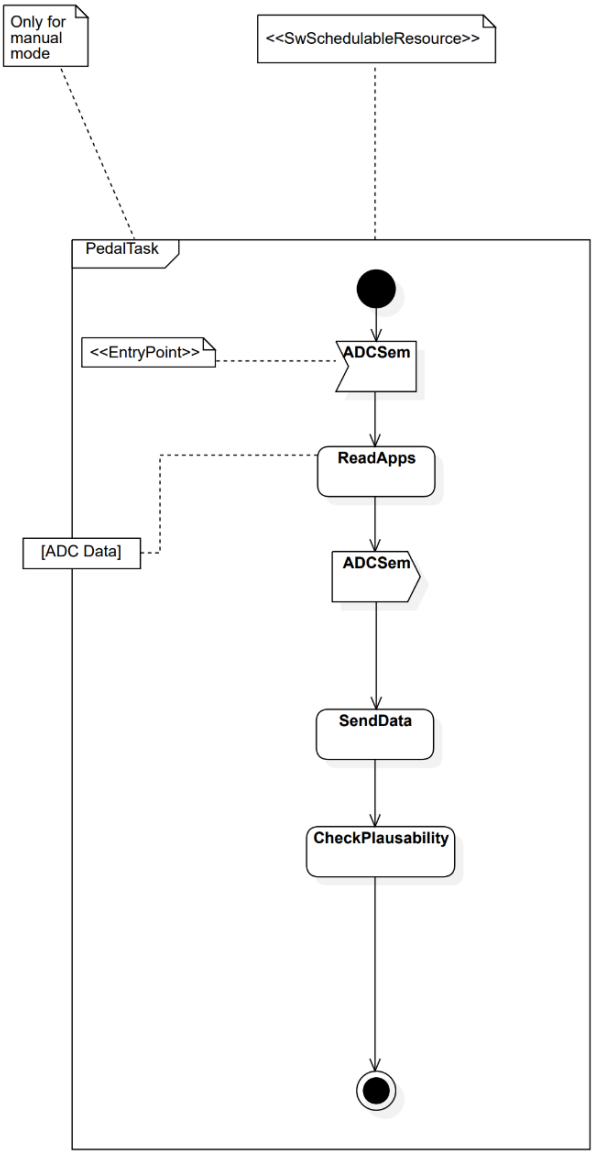

Figura 3: Pedal Task

Fan Control Task

Questo task si occupa di gestire le ventole del radiatore e dell'intercooler tramite modulazione PWM. Bloccando il semaforo della CAN motore vengono lette entrambe le temperature; seguendo quindi una certa strategia, come ad esempio una rampa, va a impostare il PWM in uscita. Il tutto verrà gestito nel modo più efficiente possibile perché le ventole hanno un consumo energetico elevato che potrebbe portare all'esaurimento della batteria se utilizzate in maniera eccessiva.

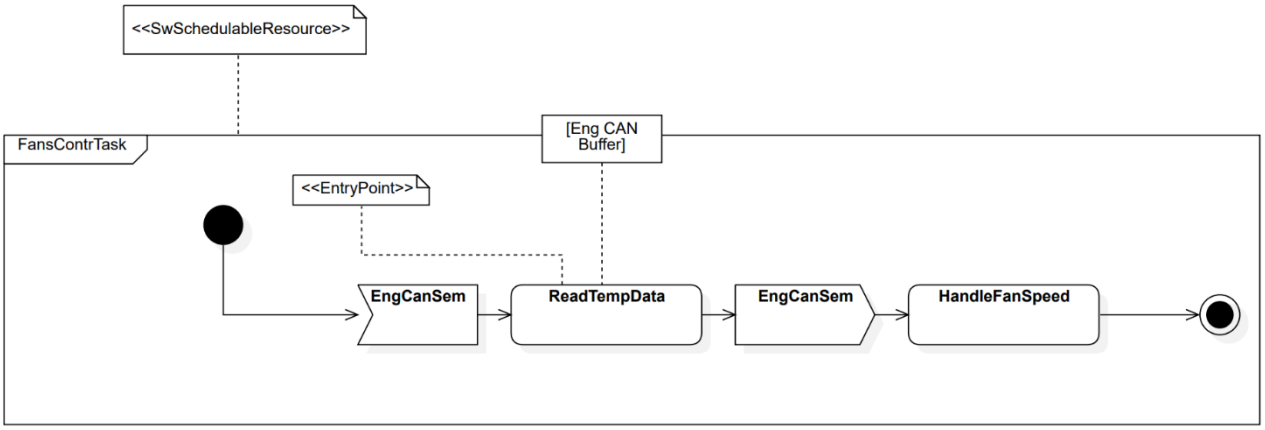

Figura 4: Fan Control Task

Telemetry Task

Si tratta del task che permette di salvare ed inviare tutti i dati di telemetria; tramite protocollo LoRa è possibile inviare i dati ad una stazione remota, quindi è possibile conoscere il comportamento della vettura in tempo reale, evidenziando eventuali problematiche attraverso la sensoristica. I dati vengono salvati anche nella vettura avendo a disposizione un logger AIM Evo 5.

La telemetria è composta principalmente dalle letture dei sensori, quindi degli ADC, ma anche i dati del sistema autonomo nel caso di modalità driverless, tra i quali lo stato ed altri valori di diagnostica. Se fosse necessario, si potrebbero inviare anche altri dati riguardanti, per esempio, il motore a combustione oppure le sospensioni che non sono lette direttamente dalla CORE, ma collegate al nostro logger AIM Evo 5.

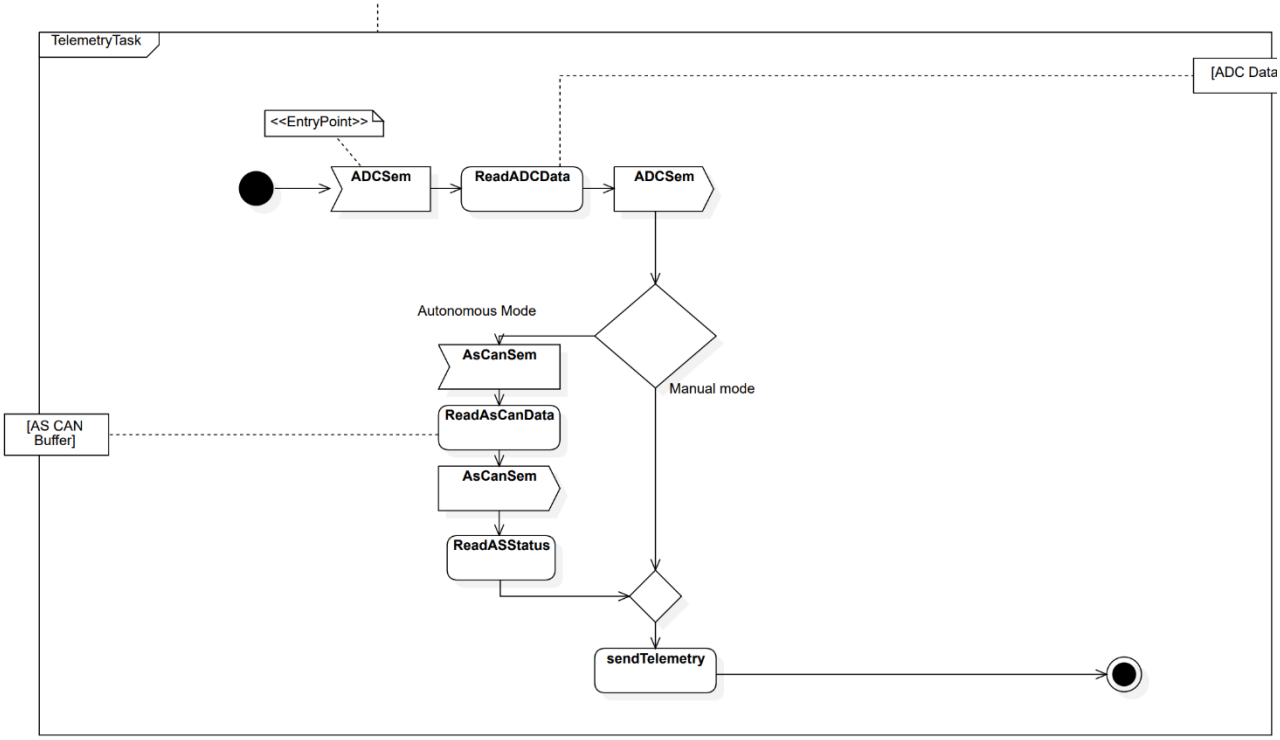

Figura 5: Telemetry Task

Gear Task

Si tratta del task che gestisce il cambio del motore a combustione interna. Per prima cosa viene sempre rilevata la marcia corrente tramite lettura ADC e semaforo, che viene subito mostrata al pilota a schermo. Viene controllato quindi se il pilota ha richiesto un cambio marcia, viene letto il numero di giri e poi vengono azionati gli attuatori per effettuare la cambiata. La lettura del numero di giri si rende necessaria ad ogni cambiata perché in corrispondenza di un basso numero di giri non possiamo effettuare un'operazione di quick-shift, ovvero la cambiata rapida senza utilizzare la frizione. Sono quindi differenziate due modalità di cambiata, una con e l'altra senza l'utilizzo della frizione.

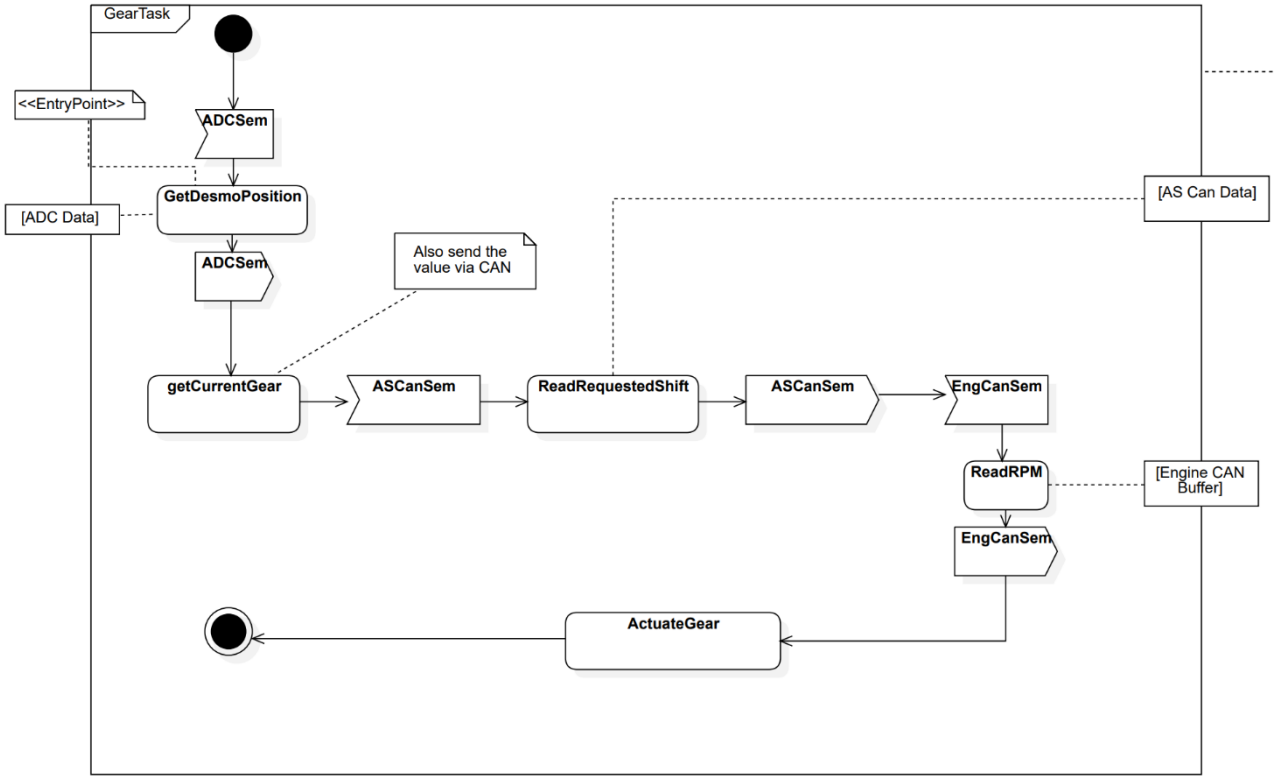

Figura 6: Gear Task

ADC Task

Questo task, semplice quanto fondamentale, ci permette di leggere gli ADC in ingresso al microcontrollore, eseguendo un polling sui dati convertiti in continuazione (Continuous Conversion Mode). L'ADC converte sempre i dati in ingresso e li memorizza in un buffer circolare; i dati vecchi vengono rimpiazzati continuamente da quelli più nuovi che vengono acquisiti tramite polling. In questo modo si crea un semplice buffer al quale tutti i task possono accedere tramite semaforo, senza creare problemi di conflitto di accesso concorrente. L'utilizzo di altre modalità quale il polling diretto, l'interrupt o il DMA con interrupt avrebbe rallentato notevolmente l'esecuzione del codice o causato cambi di contesto superflui.

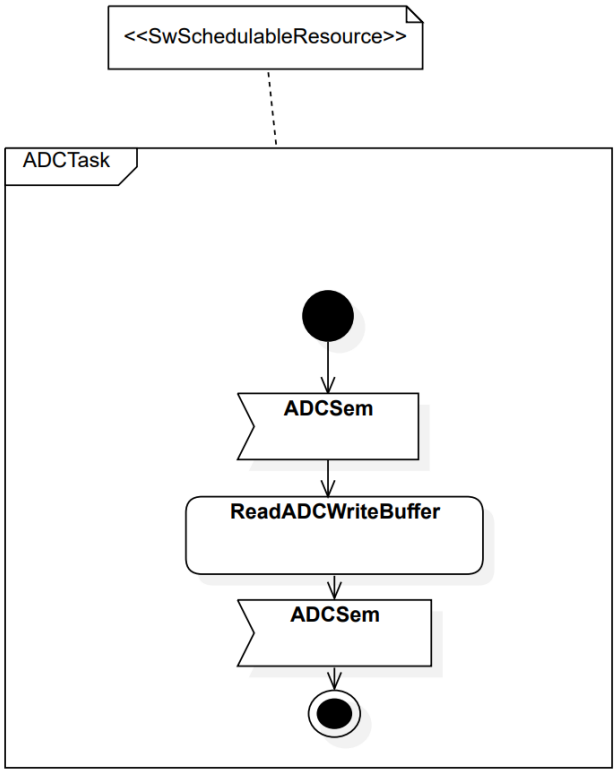

Figura 7: ADC Task

CAN Handler Task

Si tratta del task che gestisce la comunicazione di entrambe le linee CAN. Anche in questo caso, si è scelto di non utilizzare l'interrupt per avere un codice più deterministico e per evitare anche un sovraccarico del processore, nel caso ci fossero una grande quantità di messaggi in arrivo. Il task gestisce in autonomia le code in ingresso ed in uscita tramite semafori, in modo da evitare situazioni di conflitto. Avendo tanti messaggi da leggere ed inviare via CAN, abbiamo scelto un periodo di soli 40 ms per questo task. Considerata inoltre la criticità dei messaggi in arrivo dal CAN bus, soprattutto in modalità Driverless, la priorità di questo task è stata innalzata. Nella figura sottostante si possono vedere i vari messaggi di ingresso/uscita della scheda.

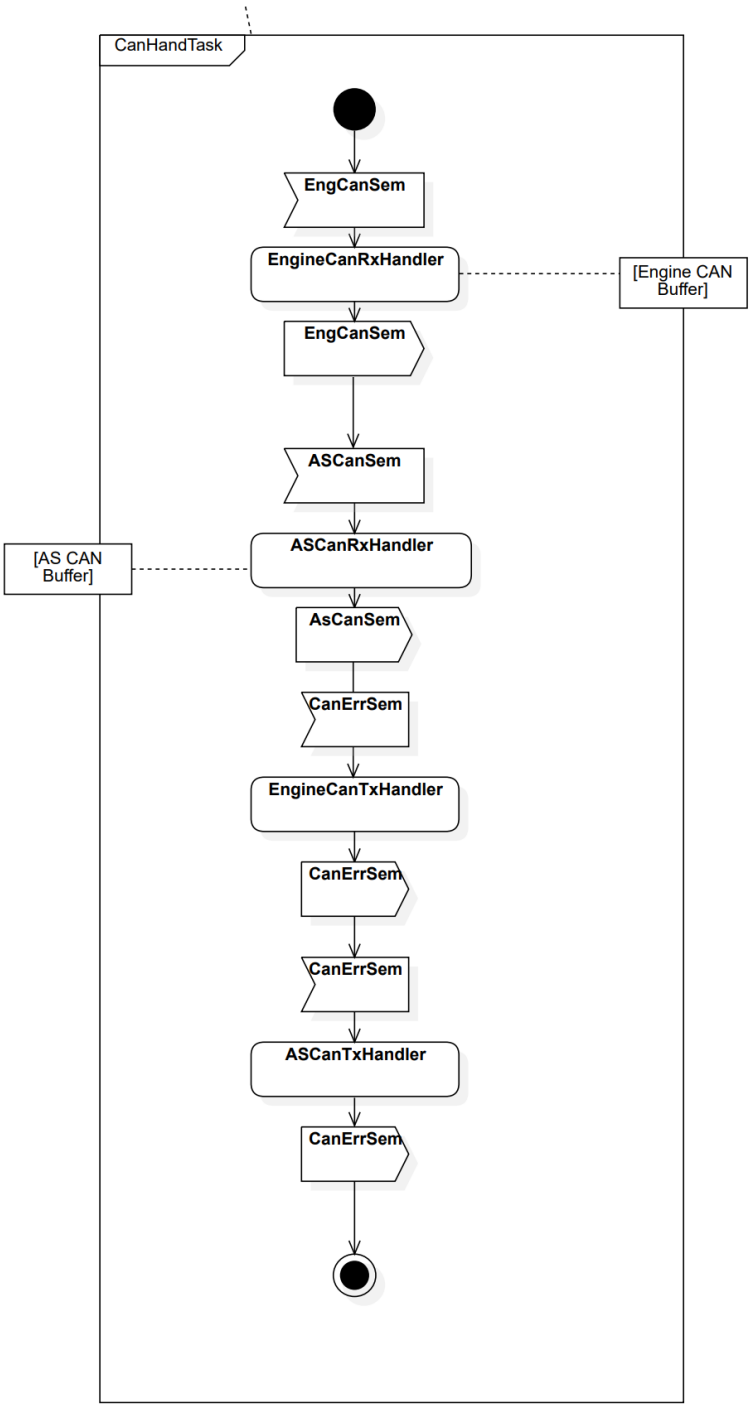

Figura 8: CAN Handler Task

AS Accelerator Task

Si tratta del task che gestisce il comando acceleratore nella modalità driverless; è infatti il PC del sistema autonomo in questo caso a fornirci l'indicazione riguardante la potenza da erogare. Il PC invia il dato via CAN e questo viene inviato alla centralina motore tramite DAC. Non essendoci il pilota umano, il task si occupa anche della gestione della frizione nelle partenze, anche quando la velocità ed il numero di giri del motore sono bassi, per mantenere il motore acceso. Il timing richiesti dai vari chunk non sono purtroppo calcolabili in quanto non è possibile esegure e testare le varie procedure senza avere la vettura a disposizione; ne è stata comunque considerata una stima all'interno delle timeline. Si suppone che il periodo di 100 ms sia più che sufficiente per garantire un corretto controllo dell'acceleratore.

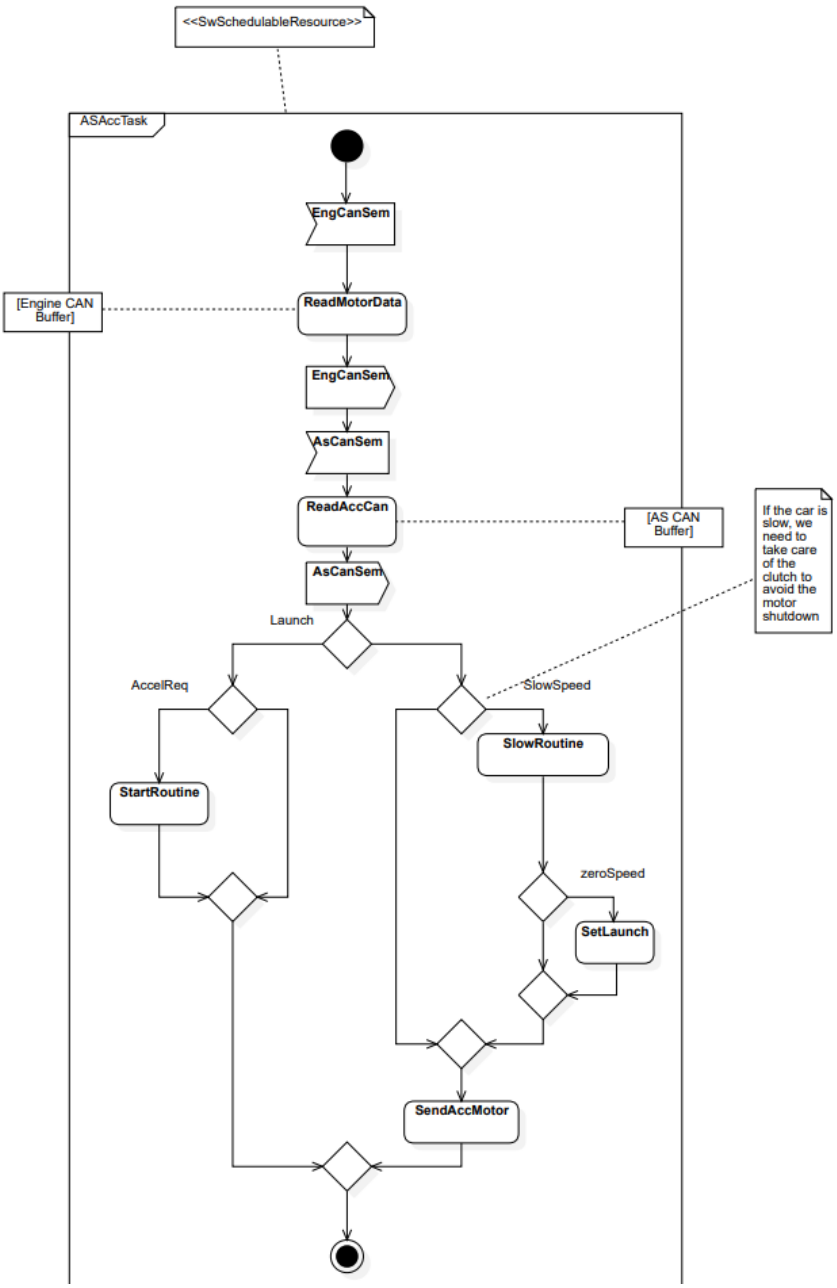

Figura 9: AS Accelerator Task

AS State Handler Task

Si tratta del task che gestisce lo stato del sistema autonomo; infatti, durante la competizione driverless, la vettura deve rispettare una serie di stati ben definiti dal regolamento. Dall'analisi dell'Activity Diagram si nota una divisione in due parti:

- **Lettura dati**, entrando nelle rispettive sezioni critiche, vengono letti e memorizzati in locale tutti i dati di interesse, quali stato del motore, stato del sistema frenante, etc.

- **Calcolo stato,** una volta raccolte tutte le informazioni, si calcola lo stato tramite una serie di if statment. Il diagramma comprende la totalità dei rami poiché è fondamentale capire come ogni singolo stato viene calcolato e cosa ciò comporta.

Si vedano in particolare gli stati:

- **ASOff**, attualmente l'intero sistema autonomo non ha ancora eseguito uno dei seguenti passaggi o non sono state fatte delle operazioni manuali:
    - attivazione dell'Autonomous System Master Switch (ASMS), che permette di accendere tutti i sistemi di attuazione.
    - controllo dello stato dell'EBS (Emergency Brake System, ovvero il sistema di frenata di emergenza)
    - la missione non è stata selezionata
    - il motore non è stato acceso

- **ASReady**, tutte le operazioni precedentemente effettuate sono state attivate. Il task a questo punto precede ad aspettare l'arrivo del Signal GO da un addetto a distanza. Questo segnale viene inviato alla vettura tramite un sistema detto Remote Emergency System (RES); il signal GO è ricevuto dalla scheda tramite un semplice GPIO. Una volta che il task rileva il signal GO , procede ad inviare all'Autonomous PC il messaggio che gli permette di prendere il controllo del veicolo.

- **ASDriving**, per essere in questo stato è sufficiente essere anche in ASReady con in più la marcia inserita, perciò è lo stato in cui si entra dopo che l'Autonomous PC prende il controllo della vettura e lì rimane fino alla fine della missione od una eventuale emergenza.

- **ASEmergency**, è stato rilevato un ingaggio del sistema di frenata di emergenza (EBS) dovuto a problemi riscontrati durante la navigazione del sistema autonomo oppure da un addetto che a distanza è in grado di attivare l'EBS tramite il sistema RES. In questo stato deve essere inoltre attivata una sirena di emergenza per 10 secondi.

- **ASFinished**, La missione è stata conclusa con successo.

Lo stato viene segnalato all'esterno della vettura tramite i LED RGB posti in modo tale da essere visibili da ogni direzione. La particolare combinazione di flash o colore, definita dal regolamento della competizione della Formula SAE, indica lo stato in tempo reale di tutto il Sistema Autonomo. Il task è periodico; ciò permette di ricalcolare lo stato del sistema autonomo ogni 200 ms.

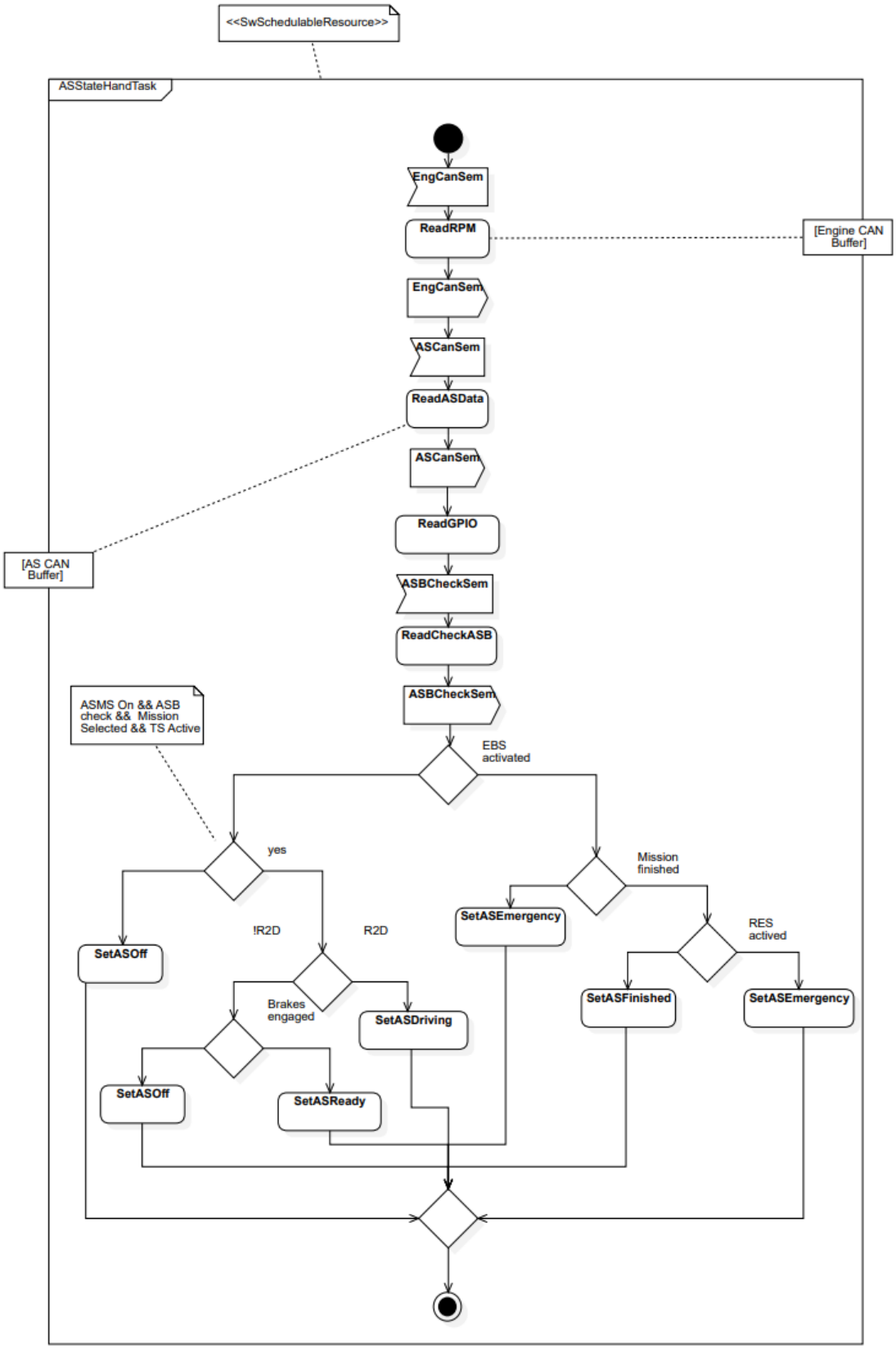

Figura 10: AS State Handler Task

AS Error Handler Task

Si tratta del task di sicurezza della vettura in grado di ingaggiare l'Emergency Brake System (EBS), ovvero il sistema di frenata di emergenza attuato tramite sistema pneumatico. Questo task controlla lo stato delle linee CAN, che sia sufficiente la pressione per una frenata nel sistema EBS e che il sistema autonomo invii un Heartbit ogni 200 ms; questi sono tutti controlli atti a garantire una risposta immediata in caso di fallimenti di una delle parti fondamentali del veicolo. Ingaggiando l'EBS viene immediatamente spento il motore e la pompa della benzina collegate allo stesso circuito detto ShutDown Circuit (SDC). Questo task ha un periodo molto basso per garantire minimo tempo di risposta in caso di fallimenti critici che potrebbero portare la vettura ad incidentarsi.

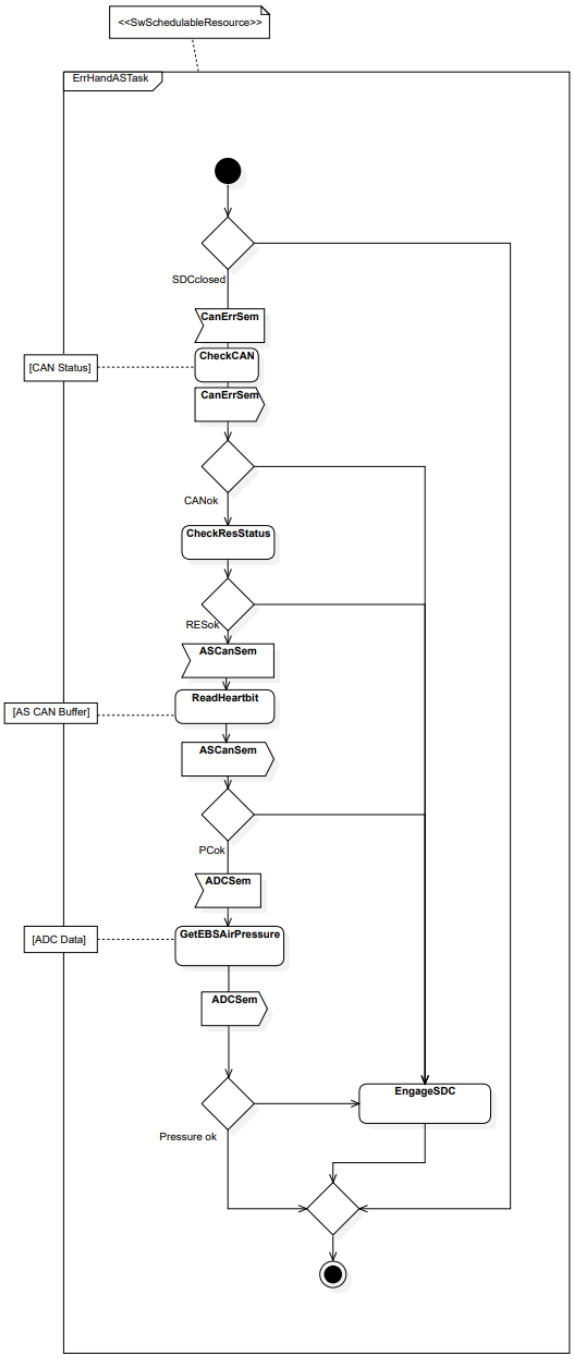

Figura 11: AS Error Handler Task

Check Mode Task

Dato che il veicolo deve comunque poter essere guidato da un pilota vero, è stato necessario costruire un task che permetta di cambiare la modalità di funzionamento della vettura: guida manuale o driverless. Questo ha ridotto il numero di tasks attivi simultaneamente, potendo disattivare i tasks della guida manuale quando in modalità driverless e viceversa, riducendo in questo modo l'overhead sul processore. Una volta scelta la modalità e, se in Autonomous, anche la missione, si provvederà ad attivare tutti i task necessari. Una volta scelta la modalità, il cambiamento non è più permesso se non riavviando la scheda; questa scelta è stata fatta per semplificare il lavoro del pilota in guida manuale. Il task prevede un periodo di 200 ms perché non è fondamentale la sua continua esecuzione, soprattutto dopo aver selezionato la modalità. Una possibile futura implementazione potrebbe essere anche la totale disattivazione del task una volta selezionata la modalità di funzionamento. ovviamente, I task attivati a seconda della modalità di funzionamento, manuale o driverless, saranno rilasciati una sola volta pur essendo questo un task periodico.

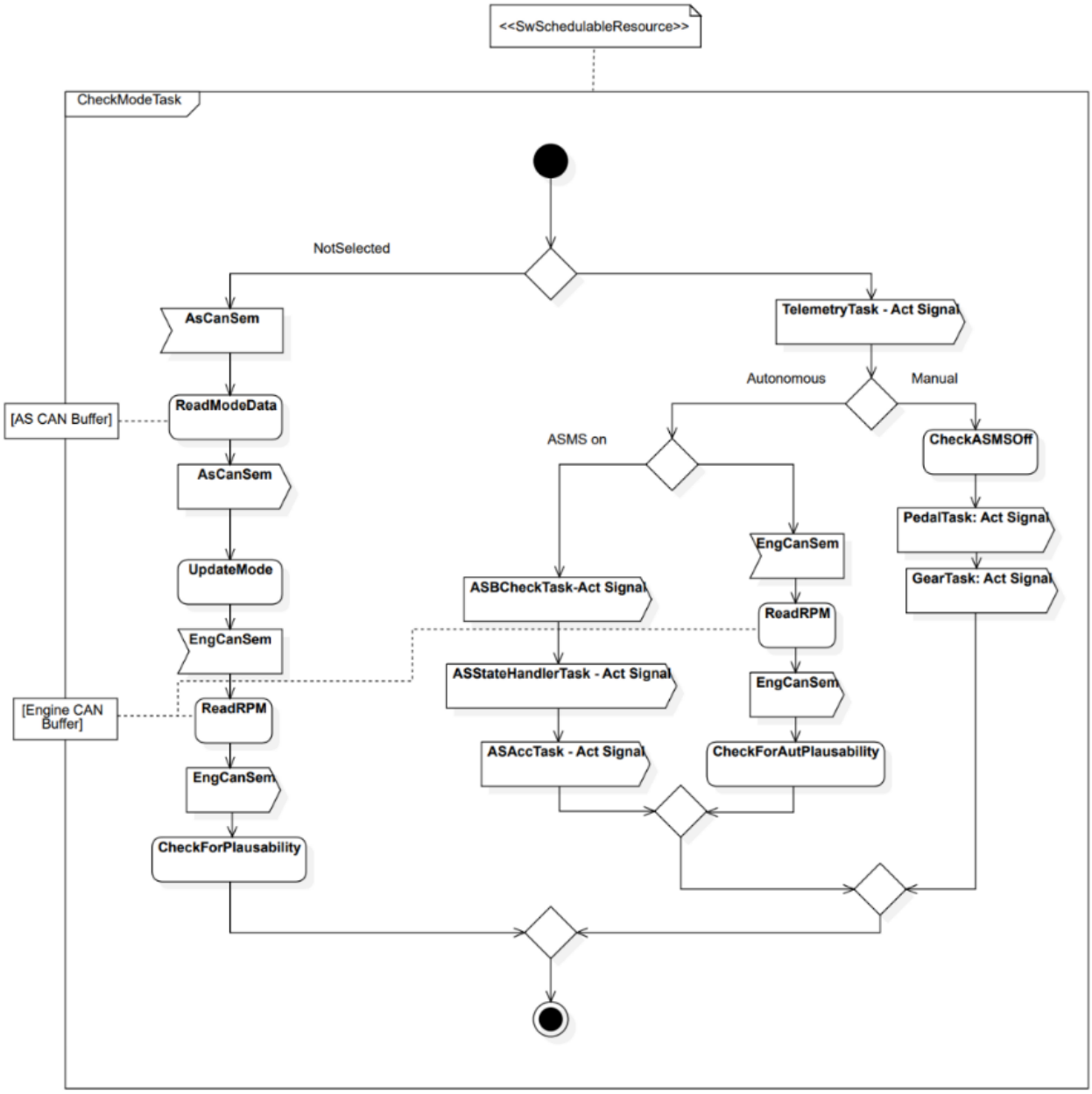

Figura 12: Check Mode Task

ASB Check Task

Si tratta del task che si occupa di fare un check iniziale dell'Autonomous System Brake (ASB). Essendo l'impianto frenante un dispositivo di sicurezza, il regolamento ci impone un controllo sequenziale di tutti i sensori di pressione riguardanti il suddetto impianto. In caso di errore, il task segnala l'anomalia e spegne il motore, non permettendo di fatto che la vettura possa procedere con le operazioni. Inoltre invia un messaggio di errore via CAN per essere visualizzato dal Display della PILOT23 e salvato dal datalogger. A procedura finita, se non ci sono stati errori, il task provvede a settare una variabile a True per indicare all'ASStateHandler Task che il sistema frenante è pienamente efficiente (in sezione critica per gestire la concorrenza di accesso al dato); inoltre provvede ad inviare al PC il primo comando di Start e successivamente viene attivato il task periodico ErrorHandAS. Il task è di tipo one-shot, quindi esegue la procedura descritta dall'Activity Diagram una sola volta. Se c'è stato un errore l'intero sistema ha bisogno di un reset e quindi il task verrà rieseguito (in caso di modalità driverless).

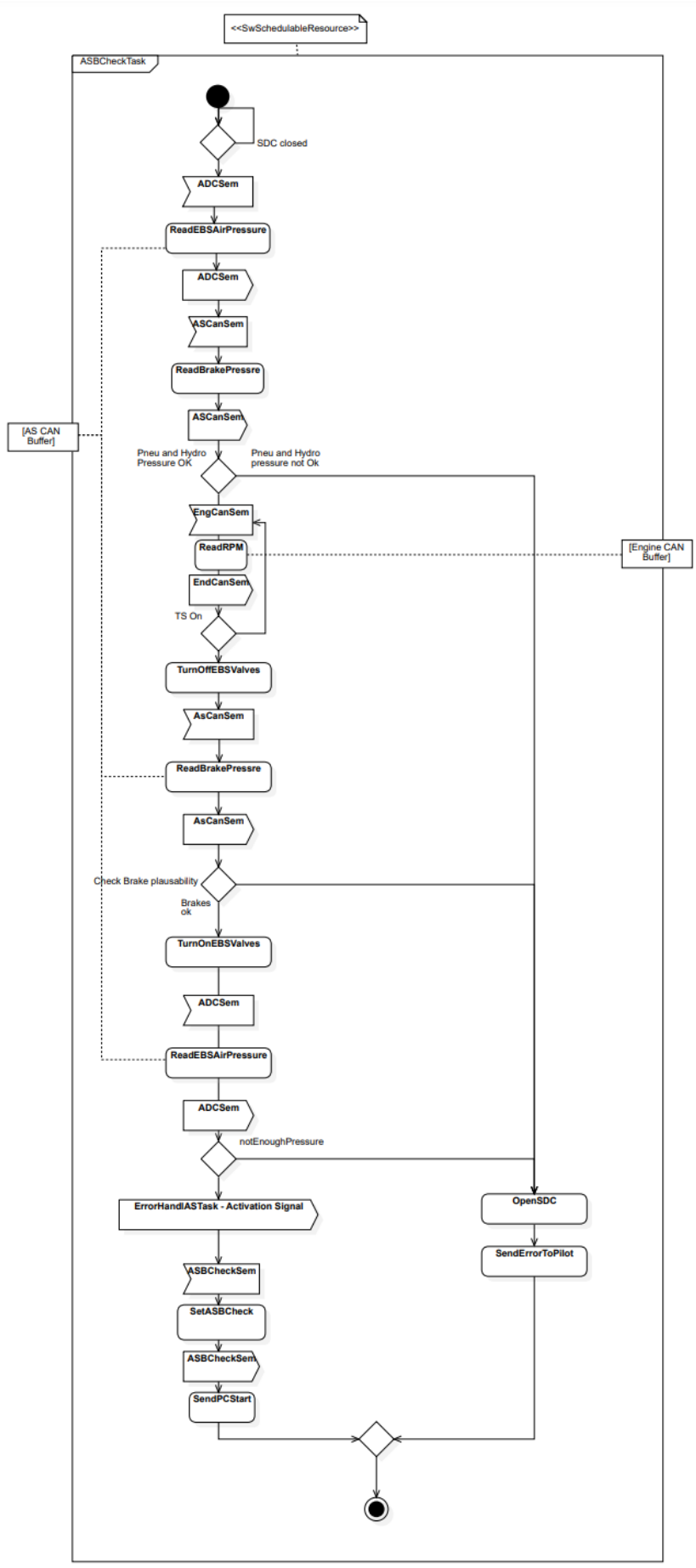

Figura 13: ASB Check Task

Timelines e PTPN

Dopo aver completato Activity Diagram e Object Diagram, è stato scritto il codice in linguaggio C tramite l'utilizzo di Visual Studio Code e STM32Cube MX. Questa toolchain è stata preferita rispetto a quella standard, ovvero l'utilizzo di STM32Cube IDE, poichè ritenuta più snella e funzionale rispetto ad un software che utilizza Eclipse.

Sono stati misurati i tempi di esecuzione dei vari task: per fare ciò, ci siamo serviti di timer hardware per avere una precisione al decimo di microsecondo. I risultati ottenuti sono stati poi riportati in microsecondi.

Ogni ET (Execution Time) è stato valutato in modo indipendente per ogni chunk dei vari task e ognuno di quest'ultimi è stato eseguito in modo indipendente per evitare prelazioni; in questo modo si ottengono dei tempi di esecuzione più realistici. Per ottenere un risultato più accurato possibile è stata fatta una media di 10000 cicli. Sorprendentemente, si è notato che i vari chunk hanno tempi di esecuzione di qualche microsecondo, vista la frequenza di clock di 216MHz del nostro microcontrollore.
Per ottenere un buon margine di sicurezza e semplificare l'analisi tramite il software ORIS, abbiamo elevato i tempi di esecuzione più brevi sino a 10 microsecondi.

Per cui possiamo già intuire la feasibility di questo taskset. I tempi delle primitive dei semafori, come quelli per eseguire le primitive per ottenere i tempi stessi, non sono stati presi in considerazione in quanto comunque risulterebbero trascurabili, considerando i vari chunk.

Ottenuti quindi i vari ET, sono state realizzate le timeline complete del taskset per ottenere così le PTPN. Entrambi i modelli sono allegati a questa documentazione.

Di seguito troviamo degli esempi, timelines e PTPN del Pedal Task e ADC Task

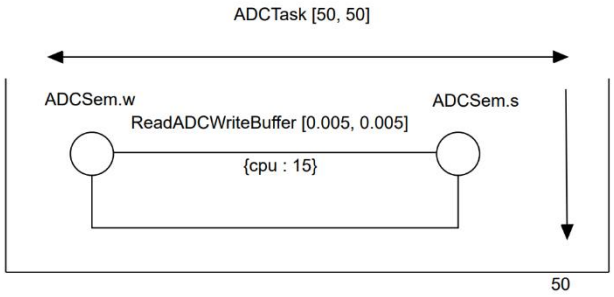

Figura 14: Timeline ADC Task

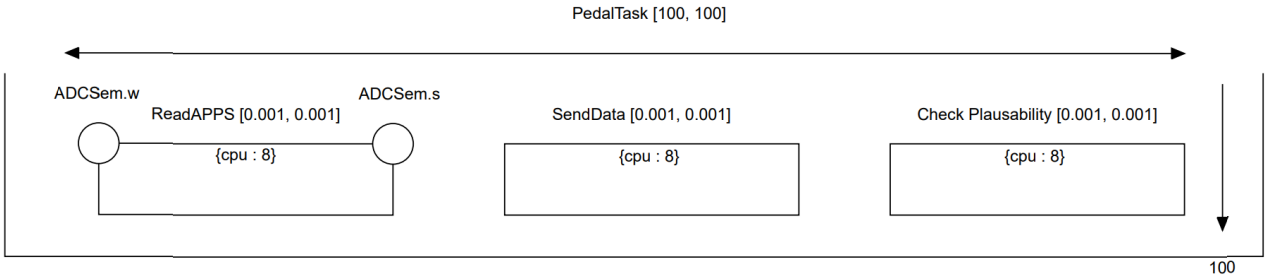

Figura 15: Timeline Pedal Task

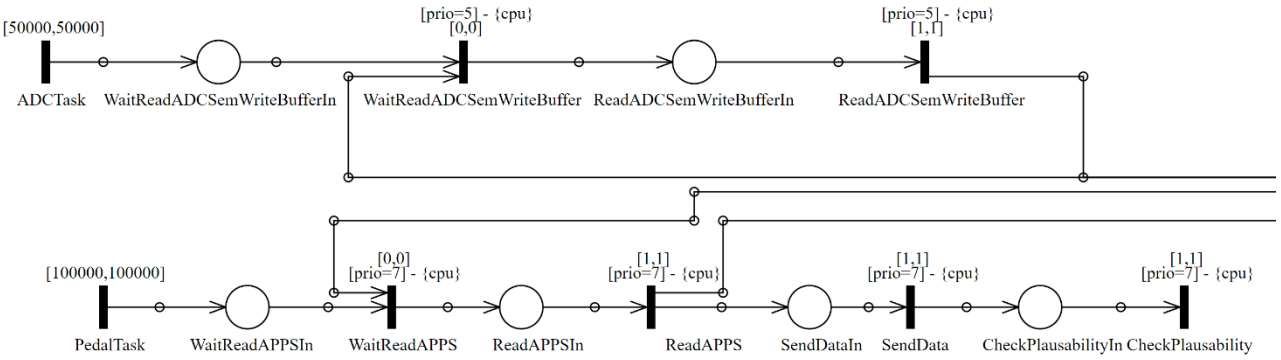

Figura 16: PTPN ADC e Pedal Task

Test e Banco Prova

Dopo aver terminato la scrittura del codice, abbiamo fatto dei test al banco prova per verificare il funzionamento effettivo dei vari task. Anche se non si tratta del veicolo reale, abbiamo collegato tutte le centraline e alcuni sensori mentre altri li abbiamo simulati. Il risultato è stato sorprendente e, dopo alcuni affinamenti, siamo riusciti a far funzionare il tutto.

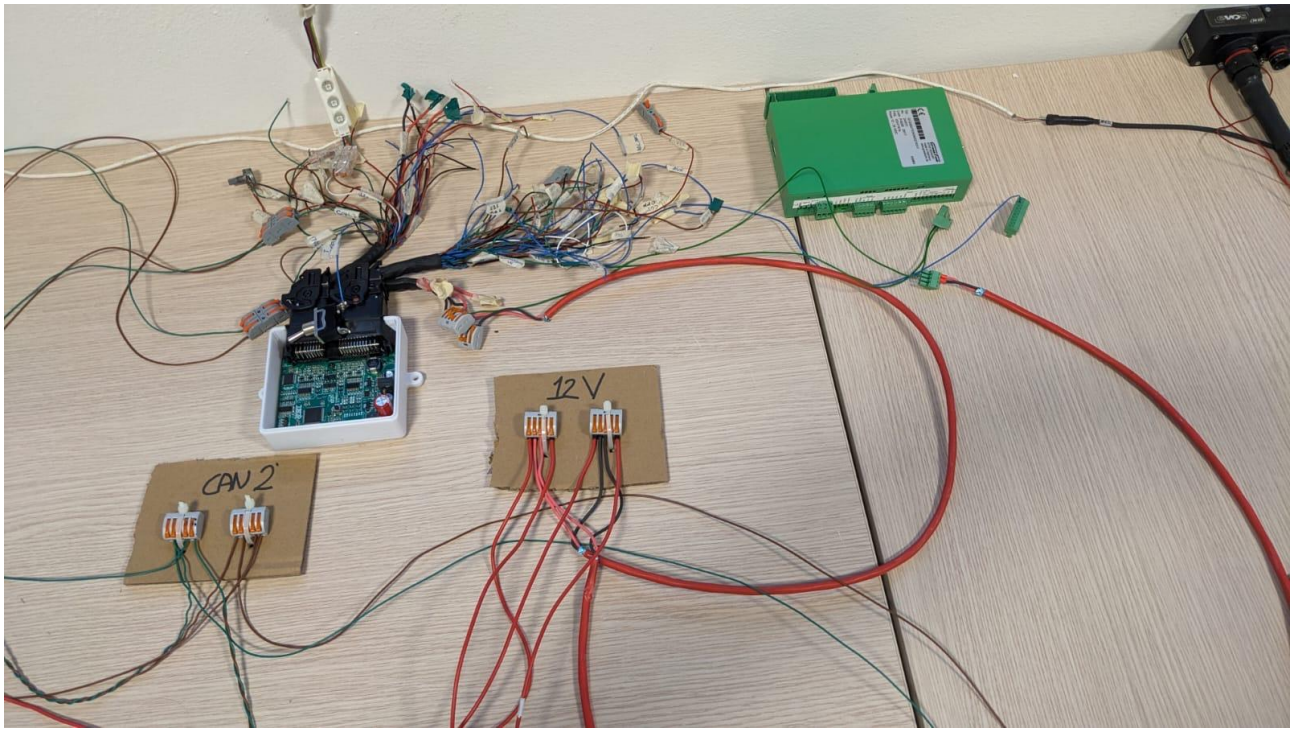

Figura 17:Banco Prova 1

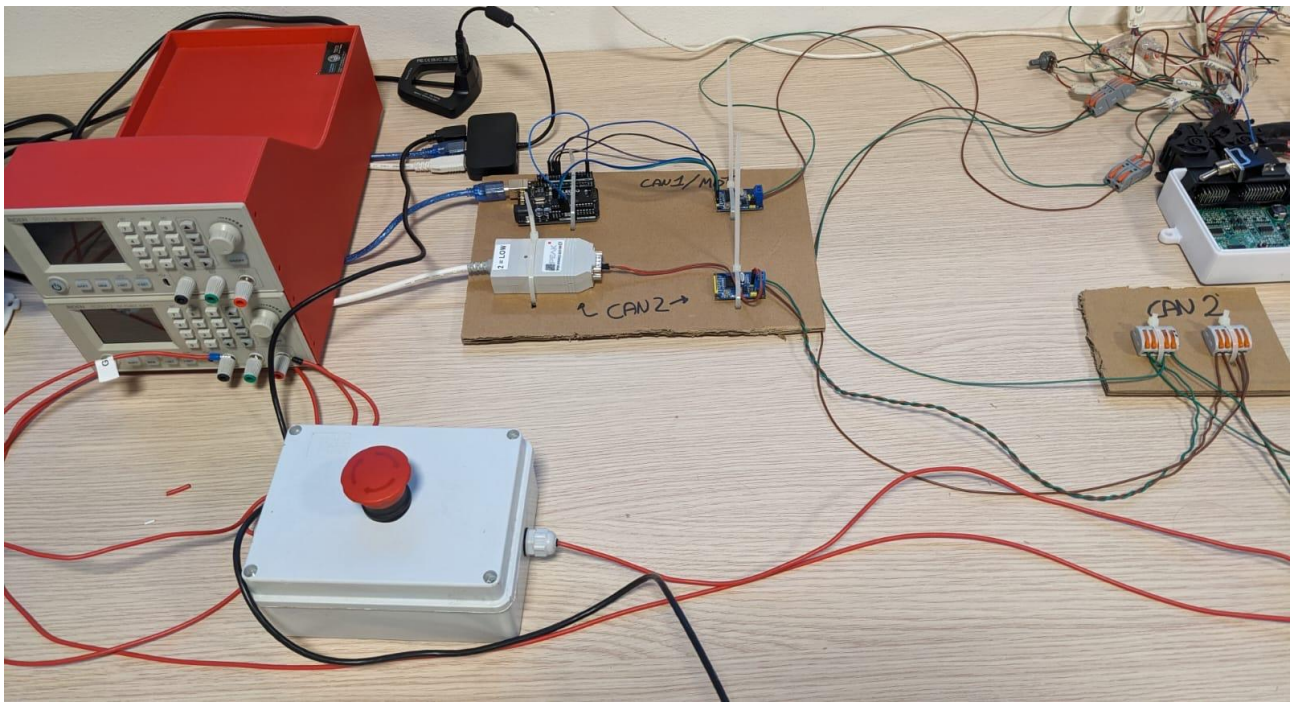

Figura 18: Banco Prova 2

Conclusioni

Arrivati alla conclusione del progetto abbiamo verificato che il codice funziona in maniera ottimale, grazie anche alle analisi e alle ottimizzazioni effettuate in fase di progettazione. Per ottimizzare il codice e renderlo più deterministico possibile, abbiamo scelto di non fare uso di interrupt. Il taskset è feasible e, se si rendesse necessario, si potrebbe aumentare la velocità di lettura degli ADC o delle linee CAN.

Il capacity factor non è elevato, ma trattandosi di un sistema hard real-time non è uno svantaggio, anzi. È fondamentale, invece, che i tempi di ripetizione, per quanto comunque relativamente lenti rispetto alla velocità del processore, vengano rispettati.

Volendo si potrebbe ridurre la frequenza di clock per diminuire il consumo energetico, anche se questo porterebbe a una riorganizzazione completa di tutti i domini di clock, compresi quelli delle periferiche. Questa operazione è quindi da effettuare con molta attenzione per evitare problematiche con timers, ADC, ecc. Sarà un compito futuro quello di ottimizzare il codice ulteriormente se si volesse procedere in questa direzione.

È stata anche sviluppata una nuova versione della CORE23, chiamata CORE24, in occasione del progetto di tesi del sottoscritto, che integra un microcontrollore STM32H7, con una frequenza di clock di 550MHz. Anche questo caso, in caso di utilizzo delle massime performance, andrebbe completamente rifatta l'analisi dei tasks oppure adeguata la frequenza di clock.