



UNIVERSITÀ DEGLI STUDI DI TRENTO

Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea in Informatica

ELABORATO FINALE

Sviluppo del Volante di
Chimera Evoluzione
Formula SAE E-Agle Trento Racing Team

Supervisore
Bruno Crispo, Paolo Bosetti

Laureando
Luca Martinelli

Anno accademico 2018/2019

Ringraziamenti

I miei Ringraziamenti vanno a tutti i membri dell'E-Agle Trento Racing Team che hanno creduto nel progetto Chimera Evoluzione e non hanno mai mollato anche quando sembrava impossibile. Ai miei genitori per essere sempre stati presenti, Alla mia famiglia (Filippo Nicola Mak e Sara) per non avermi fatto mai perdere il sorriso e aiutato a ritrovarlo quando non sembrava più possibile e Giuseppe Spallitta che ha sempre creduto in me.



Indice

Acronimi	2
Sommario	3
1 Metodologia di Sviluppo	4
1.1 GitHub e Documentazione	5
1.2 Organizzazione Team e Attività	6
2 Tecnologie	8
2.1 Scelta del Hardware	9
2.2 Scelta del Case	10
2.3 Scelta del Sistema Operativo	11
2.4 Scelta del Framework	12
3 Il Volante	13
3.1 Design della Applicazione	13
3.1.1 Interfaccia 2017	14
3.1.2 Finite State Machine	15
3.1.3 Cross Compilazione	16
3.2 Redesign della Applicazione	17
3.2.1 Dal Low Fidelity Prototype al primo Mockup	17
3.2.2 Integrazione della nuova UI del Volante	18
3.3 Funzionalità delle diverse pagine	20
3.3.1 Warning Tab	20
3.3.2 Error Tab	20
3.3.3 Status Tab	21
3.3.4 Racing Page	21
3.3.5 Battery Tab	22
3.3.6 Sensor Tab	22
3.4 Considerazioni sullo Sviluppo dell'Interfaccia	23
4 Valutazioni	24
4.1 Sviluppi Futuri	25
5 Conclusioni	27
Bibliografia	27

Acronimi

In questa sezione sono presenti gli acronimi utilizzati durante lo sviluppo della tesi.

- **AIR** Accumulator Isolation Relay
- **AP** Access Point
- **ARM** Advanced RISC Machine
- **BE** Backend
- **BMS** Battery Management System
- **CAN** Controller Area Network
- **CANH** CAN High
- **CANL** CAN Low
- **ECU** Electronic Control Unit
- **FE** Frontend
- **GPIO** General Purpose Input Output
- **HMI** Human Machine Interface
- **HV** High Voltage
- **I2C** Inter Integrated Circuit
- **IMU** Inertial Measurement Unit
- **KW** Kilo Watt
- **LV** Low Voltage
- **PCB** Printed Circuit Board
- **QFD** Quality Function Deployment
- **SAE** Society of Automotive Engineers
- **SCP** Secure CoPy
- **SD** Secure Digital
- **SSH** Secure SHell
- **UI** User Interface

Sommario



Figura 1: Foto di Chimera Evoluzione

E-Agle Trento Racing Team è un'Associazione Sportiva che nasce come progetto universitario che si occupa della progettazione, costruzione e messa in pista di un prototipo da corsa stile Formula 1 completamente elettrico. L'Associazione è nata nel Dicembre 2016 ed è costituita da studenti dell'Università di Trento dei dipartimenti di Ingegneria Industriale, Scienze Informatiche ed Economia. L'obiettivo principale di E-AGLE TRT è quello di competere su pista nelle competizioni organizzate dalla SAE International. Ad esse vi aderiscono circa 90 Università da tutto il mondo al fine di poter testare e dimostrare le proprie scelte tecniche e i propri risultati in termini di prestazioni, sostenibilità economica e marketing. Nel 2017 ho avuto l'occasione di partecipare al progetto come sviluppatore software nel gruppo Telemetria che si occupa di raccogliere e visualizzare i dati in real-time. Una parte importante del lavoro è stata condividere con gli altri membri del team le competenze di ingegneria del software per poter definire una struttura per tutti gli studenti occupati nell'ambito informatico e dare una forma alle documentazioni. Inizialmente mi sono occupato dell'implementazione di nuove funzionalità, come la calibrazione della pedaliera direttamente dal Volante, fino alla visualizzazione sull'interfaccia dei sensori presenti in macchina.

Questo ha permesso di rendere più stabili le funzioni principali per poi, nel secondo semestre del 2018, ridisegnare la digital dash del Volante per facilitarne l'utilizzo, rendendolo più accattivante e migliorandone l'aggiornabilità. Tutt'ora sono impegnato come Project Manager della parte telemetrica grazie agli ottimi risultati ottenuti lo scorso anno.

1 Metodologia di Sviluppo

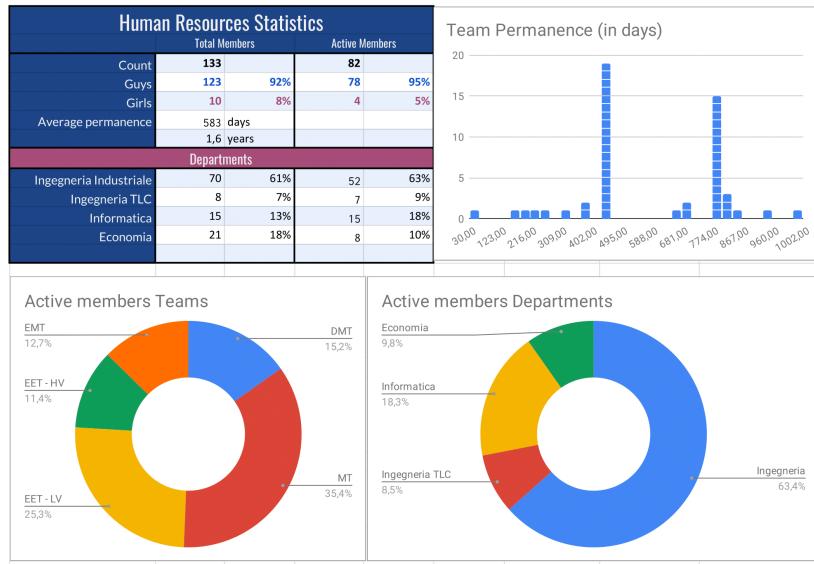


Figura 1.1: E-Agle Trento Racing Team

L'organizzazione del Team a Settembre 2017 dal punto di vista software non era ben definita, alcune parti di codice erano presenti online, altre solo mantenute in locale dalle persone interessate. Questa modalità ha avuto molti limiti, soprattutto per quanto riguarda il version controlling e l'aggiornabilità del codice, non avendo la possibilità di mantenere diverse versioni in parallelo. Per cui confrontandoci con gli altri studenti si è deciso di avere un profilo pubblico della squadra su GitHub per poter mantenere il codice secondo gli standard attuali e poterlo condividere con chi d'interesse. La scelta di GitHub è stata spontanea, dovuta ai suoi vantaggi rispetto a BitBucket per la partnership con l'università e quindi tool inclusi. Questo ha migliorato notevolmente il workflow di chi lavorava sulle varie parti della macchina e il tempo di apprendimento degli studenti che hanno deciso di avvicinarsi al progetto soprattutto grazie alla realizzazione, attraverso Sphinx e GitHub Pages, di una raccolta di documentazioni e procedure da tramandare.

1.1 GitHub e Documentazione

I due principali strumenti introdotti sono stati GitHub per il version controlling e la condivisione del codice e Sphinx per generare documentazione partendo da documentazioni che altri studenti avevano già scritto.

L'approccio a GitHub è stato lento, ma ha permesso di poter dare uno spazio a tutto il codice del Team in breve tempo. Le repository sono gestite da un admin che si occupa di aggiungere gli studenti interessati e consiglia come mantenere il codice grazie a controlli periodici. La struttura delle repository è definita in base alla monoposto e al sottogruppo di interesse, o più in generale una parte della macchina. Un esempio sono le due versioni del Volante, *chimera-steeringwheel* [1] e *fenice-steeringwheel* [2]. Per quanto riguarda la repository del Volante è stato deciso di mantenerla pubblica. Essendo un progetto universitario che vede l'impegno di più persone sotto più ambienti (software, hardware e meccanico) ci è sembrata la scelta giusta presentare l'esempio di una interfaccia unica nel suo genere che utilizza diverse tecnologie da cui le persone possono ispirarsi e utilizzarla come esempio per le librerie del framework e le tecnologie usate.

L'introduzione di Sphinx è nata dalla necessità di condividere le conoscenze con altri studenti che si avvicinano al progetto e trovare un punto raccolta unico e accessibile da esterni del materiale ritenuto utile e non sensibile. La scelta di utilizzare Sphinx come strumento è dovuta alla consultazione, in quel periodo, di alcune documentazioni relative a schede programmabili con Python. Questa documentazione mi è sembrata fin da subito semplice e facilmente mantenibile da parte di tutti. Una delle prime guide pubblicate è stata "*Come cross-compilare per Raspberry da Linux*" per il framework Qt, siamo partiti da una guida trovata su Medium [5] per arrivare alla nostra versione per il Team Telemetria. La scelta di utilizzare Sphinx come generatore di documentazione ha mostrato un grande vantaggio quando è stato deciso di rendere pubblico il materiale della Design Presentation.



Figura 1.2: Sito Documentazione del Team [6]

Questi materiali sono stati portati dal formato *.docx* a *.rst*, utilizzato da Python per creare pagine *.html* statiche. Sono state poi rese disponibili online e accessibili attraverso QR code dai nostri rollup che esponiamo durante gli eventi. Il risultato è stato quello di poter mostrare agli sponsor e alle persone interessate al progetto il lavoro svolto per ogni componente realizzata da noi e le scelte tecniche fatte.

1.2 Organizzazione Team e Attività

Durante le riunioni con il gruppo telemetria ho riscontrato delle difficoltà da parte dei membri nell'organizzazione del lavoro e nell'avere chiari gli step necessari per il raggiungimento dell'obiettivo principale. Per ovviare a questo problema si è rivelato molto utile un approccio *agile* e l'utilizzo di tavole *kanban*.

Predisporre il team all'utilizzo di metodi agili ci ha permesso di ridurre il rischio di fallimento durante lo sviluppo, testando in locale e poi portando in macchina la funzionalità senza riscontrare errori. Il problema principale che emerge dai partecipanti al progetto è quello di non poter dedicare il tempo che desiderano al team, avere un approccio *agile* ci ha permesso di sviluppare in breve tempo e in modo ordinato tutto quello che ci veniva richiesto venendo incontro alle esigenze dei singoli.

Il risultato è stato un approccio diviso in queste fasi:

- Pianificazione: Determinare, concordando con i membri del team attraverso riunioni e/o resoconti dei test, quale funzionalità deve essere implementata.
- Analisi dei requisiti: Come può essere integrata la funzionalità all'interno del nostro codice, valutando librerie e tecniche alternative.
- Progettazione: Divisa in due parti, la prima, molto importante nel nostro caso, disegno su carta per poi passare alla seconda che riguarda la stesura del codice.
- Test: Svolti principalmente in locale e poi portati sul dispositivo senza riscontrare problemi di compatibilità.

Per quanto riguarda il Volante inteso come "*prodotto finito*" i passi necessari sono stati più articolati. Tutti i membri del gruppo si sono occupati di raccogliere delle possibili soluzioni per poi essere discusse e confrontate anche con chi non era coinvolto direttamente, per poter condividere il più possibile il *know-how*. Attraverso riunioni periodiche e la raccolta di feedback sulla nostra monoposto è stato possibile, grazie alla disponibilità delle persone coinvolte, di avere sempre chiaro cosa andava e cosa no. Questo è stato molto importante perché ha definito una tipologia di task specifica che ogni membro del team, non per forza coinvolto nello sviluppo software, doveva portare a termine.

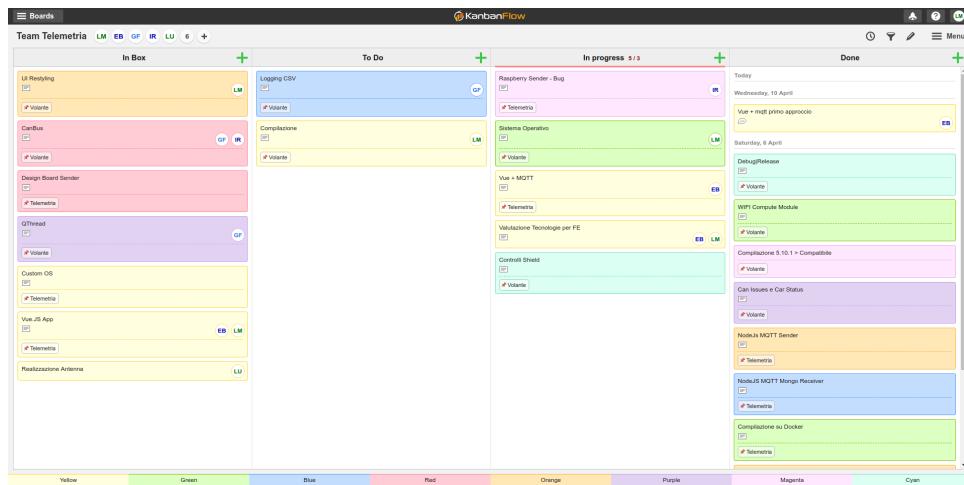


Figura 1.3: Tavola Kanban utilizzata dal Team Telemetria

L'introduzione di Kanban ha migliorato il flusso di lavoro, aumentato la produttività e la qualità del prodotto finale in modo considerevole. Kanban fa parte dei metodi agili e come tale rende il processo lavorativo molto più flessibile, non sempre le task sono state portate a termine nelle scadenze prestabilite, questo per impegni universitari e mancanza di esperienza nello stimare i tempi. I compiti sono stati suddivisi in piccole task e portati a termine l'uno dopo l'altro. Tuttavia data la sua flessibilità abbiamo sempre mantenuto indipendente lo sviluppo di una parte rispetto ad un'altra e grazie alla facilità di creare prototipi in grado di testare le singole funzionalità in poco tempo non si sono mai

trovate delle situazioni in cui non potevamo testare perché non avevamo un finito una task. Lo sviluppo in parallelo, considerato come attività multitasking, è stato sempre mantenuto in quanto le competenze dei singoli membri del team erano sempre abbastanza specifiche.

Il team che si è occupato dello sviluppo del Volante e della sua realizzazione è composto da:

- Davide Farina: Pilota e supervisore del progetto
- Luca Martinelli: Main Developer
- Daniele Faccinelli: PCB Designer
- Laura Scoccianti: UI Designer
- Ciro Malacarne: Case Designer
- Damiano Iob: Responsabile del Sistema Operativo

Ogni parte dello sviluppo è stata messa su Gantt e poi, per avere una visione specifica del lavoro, le fasi del Gantt sono state espanso e analizzate come tante singole task identificando poi la fine come una *Milestone*. Nel 2019 come strumento è stato utilizzato *waffle.io*, in quanto è facilmente integrabile con la repository GitHub. Quest'anno, invece, causa la chiusura del sito, ci siamo spostati a *Kanban Flow* perché non avendo esigenze particolari qualsiasi applicazione web poteva fare al caso nostro. L'unico requisito è stato la semplicità d'uso e avere più progetti nello stesso tempo. Gli studenti coinvolti, anche se non direttamente interessati allo sviluppo software, utilizzavano la tavola kanban per avere sempre sotto mano lo stato dei lavori. Per quanto riguarda la creazione di nuove task, e più in generale, la sua gestione, se ne è occupato l'amministratore.

Una considerazione importante da fare è che questi problemi sono emersi durante la mia presenza nel team e precedentemente non erano mai stati rilevati. Lo scopo principale fin da subito è stato la realizzazione della macchina e mentre c'è stata attenzione per l'organizzazione in senso generale si è trascurato l'aspetto organizzativo del singolo sotto-team e in particolare dello sviluppo software. Le motivazioni sono da trovare nel breve tempo che si ha a disposizione per sviluppare e testare e nell'inesperienza delle persone coinvolte visto che il progetto è nato da poco e dobbiamo confrontarci con studenti di facoltà che hanno squadre con esperienza di oltre 10 anni di competizione. L'introduzione di queste tecnologie e la formazione delle persone per approcciarsi ad esse ha fatto sì che le scelte tecniche potessero essere prese con più consapevolezza, alle volte in modo indipendente, grazie alla divisione dei compiti in base alle aree di competenza. Scelte che si sono poi rivelate vincenti nel corso del progetto.

2 Tecnologie



Figura 2.1: Esploso del Volante

Durante l'intero progetto del Volante si è sempre cercato di confrontarsi con professori competenti nel settore di referimento. Questo ha portato ad acquisire sempre più idee possibili per poter trovare soluzioni ottimali e innovative ai nostri problemi. L'interdisciplinarietà del progetto ha fatto sì che professori, provenienti anche da dipartimenti diversi avessero la possibilità di dare importanti contributi e utili consigli, su quali tecnologie potessero fare al caso nostro. Questo ha fatto sì che ogni singola scelta dovesse essere motivata e costruita da una base di esperienze personali facoltative e considerazioni giustificate con esempi. Queste scelte sono poi state discusse in gruppo per essere approvate da tutti. Le scelte prese sono state abbastanza indipendenti, considerando il settore meccanico e software/elettronico, considerando le limitazioni che si potrebbero affrontare. Non si sono verificate situazioni in cui una nostra possibile scelta non potesse essere presa vedendo come ostacolo ad esempio il case del Volante. Le difficoltà invece si sono ritrovate nel interfacciare il sistema operativo con l'hardware che, per nostra inesperienza, ha portato ad alcune situazioni in cui i lavori sono rallentati.

2.1 Scelta del Hardware

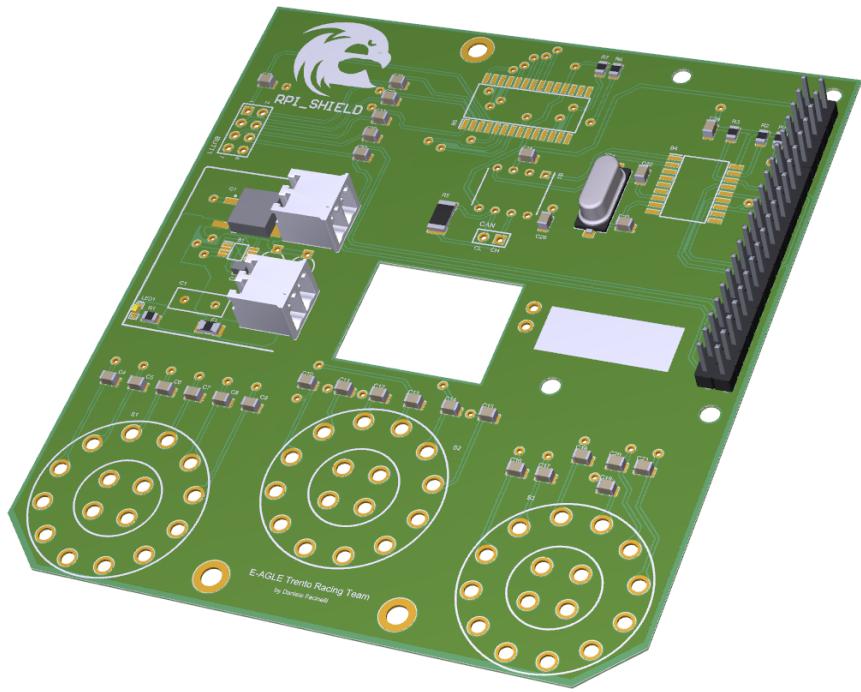


Figura 2.2: Shield del Volante

La scelta della scheda si è basata principalmente su due aspetti molto importanti, il prezzo e la facilità di sviluppo. Dopo aver fatto ricerche per la compatibilità di diversi software l'utilizzo di un Raspberry è risultato sin da subito la scelta più pertinente dato il nostro background di studenti appassionati di elettronica e software. In particolare la nostra attenzione è caduta su Rasberry Pi Zero un dispositivo low cost, con la possibilità di avere una connessione wifi on-board (facilità di collegarsi ad AP), dimensioni e peso ridotte e adattabile a diversi dispositivi di input/output. Raspberry vanta una community di appassionati e sviluppatori nel settore informatico molto vasta, questo ha permesso di trovare esempi e documentazione con facilità. Raspberry Pi Zero può essere poi aggiornato alle versioni più performanti senza troppi problemi, montando un processore ARM prodotto dalla casa Broadcom, nel nostro caso (BCM2835), necessitando solo di aggiustamenti software, mantenendo uguale l'ambiente software. Ciò che rende unico il nostro Volante è il fatto di essere custom in tutte le sue parti. Per necessità è stata disegnata una shield collegata ai GPIO del Raspberry per poter aumentare e utilizzare al meglio le sue funzioni di input e output. Per quanto riguarda le interfacce di input del pilota, per poter utilizzare più tasti e più manettini (3 manettini da 6 posizioni), è stato introdotto sulla shield un I2C multiplexer, precedentemente utilizzando un solo manettino non era necessario. Come display è stato scelto di utilizzare un LCD 4.3" TFT, collegato alla scheda via HDMI, grazie a un modulo di *Adafruit* per poter avere una maggiore scelta, in quanto sul mercato momentaneamente per quanto riguarda le specifiche tecniche i display che escono direttamente in HDMI hanno una luminosità troppo bassa per poter essere usati sotto il sole. Le nostre necessità sono state, dal punto di vista del pilota, di avere quattro tasti fisici nella parte frontale per poter utilizzare l'interfaccia, due paddle per potersi muovere nelle diverse tab e tre manettini per poter gestire la configurazione della macchina. Per quanto riguarda invece l'interfaccia di comunicazione con il resto della macchina si è reso necessario l'utilizzo del SPI per poter comunicare in Can-Bus utilizzando l'MCP2515 e l'alimentazione, che passa per il sistema di aggancio rapido, per poter alimentare il Volante a 5V.

2.2 Scelta del Case



Figura 2.3: Primo Mockup del Volante di Chimera Evoluzione

La scelta di ridisegnare il case, rispetto al modello dell'anno precedente nasce dall'esigenza di avere una maggiore ergonomia, una maggiore usabilità e la possibilità di aggiornare le funzionalità in corso di sviluppo. I materiali utilizzati nella versione precedente non erano adatti ad un uso sotto stress e non risultava facile il debug della componentistica hardware in fase test. Per quanto riguarda il design del Volante di Chimera Evoluzione le scelte sono state prese per soddisfare il regolamento e valutando lo spazio all'interno riservato alla componentistica elettronica. I requisti che abbiamo considerato sono stati una maggiore robustezza, ergonomia qualità dei materiali, facilità di assemblaggio ed estetica per venire incontro alle esigenze del pilota. Per la parte di produzione è stata fatta una ricerca sulle migliori soluzioni che potevamo utilizzare per costruirlo e con i materiali che più venivano incontro alle nostre esigenze.

Tutte le componenti sono state disegnate e realizzate dal team nei nostri laboratori.

- Piastra Principale: fibra di carbonio con core interno in schiuma da 1.5 mm per uno spessore totale di 3 mm, trattamento di cura in autoclave. Rifinito con tecnologia di taglio ad acqua per ottenere la forma desiderata.
- Paddle: 5 strati di fibra di carbonio, Trattamento di cura in autoclave, rifinito con taglio ad acqua.
- Mascherina Frontale: Stampata in 3D con processo di stereolitografia, Materiale: Resina, Stampante: Form 2
- Manettini: MJF (Multi Jet Fusion) with HP Jet Fusion 3D 4200 printer, PA 12 powder, sandblasted and painted
- Impugnatura: Stampata in 3D con processo di stereolitografia, Materiale: Resina, Stampante: Form 2. Incollati con adesivo strutturale.
- Guscio Posteriore: Alluminio serie aeronautica, lavorato da un blocco grezzo con fresa CNC a 5 assi

2.3 Scelta del Sistema Operativo

```
git clone https://github.com/element14/raspbian-build-tools.git
cd raspbian-build-tools
./build_rpi.sh -t buster

# Build RPi OS
cd /home/pi/Desktop/buster
./build_rpi.sh -t buster

# Copy RPi OS to SD card
cd /home/pi/Desktop/buster
./copy_sdcard.sh -t buster

# Boot RPi OS
cd /home/pi/Desktop/buster
./boot_pi.sh -t buster
```

Figura 2.4: Yocto preparazione dell'ambiente

La scelta della scheda ci ha portato ad utilizzare in fase di test una delle ultime versioni stretch di Raspbian presenti nel 2017. Le ottimizzazioni sono state in termini di boot time, l'autostart della applicazione, la connessione a reti wireless, la configurazione dell'uscita hdmi per il display e le librerie necessarie per poter eseguire il programma. I nostri requisiti sono stati principalmente due, il poter compilare senza dover utilizzare il Volante, lasciando così il dispositivo il più leggero possibile e poterlo utilizzare come sistema di debug della macchina. L'utilizzo di *SocketCan*, attraverso il pacchetto *can-utils* ci ha permesso in più situazioni di gestire e leggere i messaggi senza eseguire il software del Volante, ma come una macchina linux collegata ad essa via *ssh* e per aggiornare il codice via *scp*. Una volta definiti tutti i requisiti e attraverso il progetto open-source *buildroot* abbiamo sviluppato un sistema operativo con tutte le specifiche richieste, successivamente le condizioni sono cambiate e abbiamo deciso di spostarci verso un altro ambiente di cui è presente più documentazione che spiegherà nel prossimo sottocapitolo. Grazie alla diversificazione delle ricette è possibile aggiornare la componentistica hardware cambiando alcuni parametri prima della fase di build, questo ci ha permesso di mantenere una buona stabilità nel workflow senza dover mantenere più progetti, ma solamente di modificare la ricetta di quello già esistente e integrare i pacchetti necessari per la scheda di riferimento. Mantenere lo stack di un sistema operativo Linux ha permesso di integrare facilmente le funzionalità dalla versione desktop a quella in macchina, soprattutto per la cross-compilazione. Durante il secondo semestre del 2018 per necessità del team mi è stato chiesto di verificare alcune soluzioni per poter avere un sistema operativo "*real-time*", tra le possibilità la più interessante sulla quale ho fatto alcuni test e ricerche è risultata essere *Xenomai*, ma per mancanza di tempo e sviluppatori il progetto è stato abbandonato dovendo concludere lo sviluppo del Volante per Chimera Evoluzione. Successivamente, grazie ai risultati ottenuti con il Volante, questo non è risultato più un problema avendo altre alternative più facili da implementare.

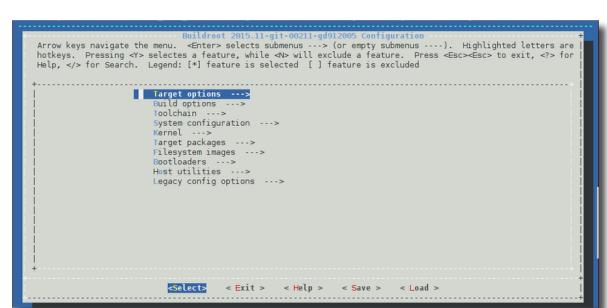


Figura 2.5: Builroot - Primi passi della fase di build

2.4 Scelta del Framework

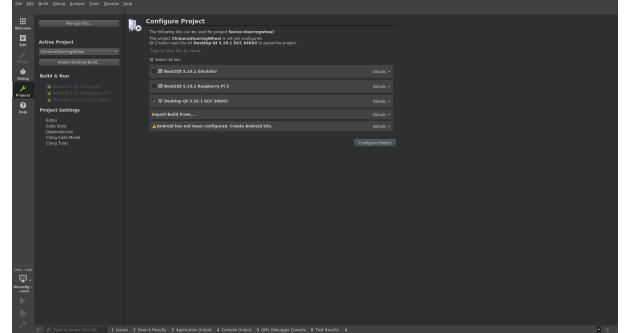


Figura 2.6: Qt Creator - apertura di un progetto e scelta del compilatore

La scelta del framework era già stata fatta precedentemente il mio arrivo nel team, la decisione che è stata presa per Chimera Evoluzione è stata quella di voler terminare il lavoro dell'anno precedente aggiornarlo e renderlo il più possibile stabile e adatto alle nostre esigenze. Il framework in considerazione risulta essere Qt, che permette grazie alle sue librerie grafiche multipiattaforma di lavorare in C++ e disegnare in breve tempo interfacce grafiche. Qt, oltre ad essere utilizzato per soluzioni embedded, vanta numerosi esempi nel campo dell'automotive, è infatti presente nelle dash digitali di alcuni modelli nel mercato automobilistico. I vantaggi di questo framework sono la diversificazione del lavoro in termini di sviluppo, chi si occupa di aggiornare la parte grafica, quindi il FE, non per forza deve essere in stretta relazione con chi sviluppa la parte di backend del progetto, garantendo un workflow più dinamico. Qt nasce come framework open-source, i progetti che abbiamo preso in mano per capirne le potenzialità erano sviluppati su Raspberry o comunque sistemi Linux e facilmente portabili a piattaforme embedded, non tanto per la parte di compilazione ma del progetto in senso stretto. Inoltre Qt risulta abbastanza intuitivo al primo approccio utilizzando la sintassi di C++ ma, le prime difficoltà si riscontrano nell'utilizzare le funzioni proprietarie che, anche se fornite con una buona documentazione, non sempre risulta completa.

Questo framework presenta anche una versione commerciale distribuita da *The Qt Company*, con la possibilità di utilizzare *tool* proprietari disegnati per facilitare il lavoro degli sviluppatori e ampliarne le funzionalità attraverso ulteriori librerie e pacchetti per compilare su dispositivi embedded. Oltre alla community, che presenta sempre discussioni interessanti sulle nuove funzionalità implementate, con la versione commerciale si ha a disposizione un servizio di assistenza e consulenza.

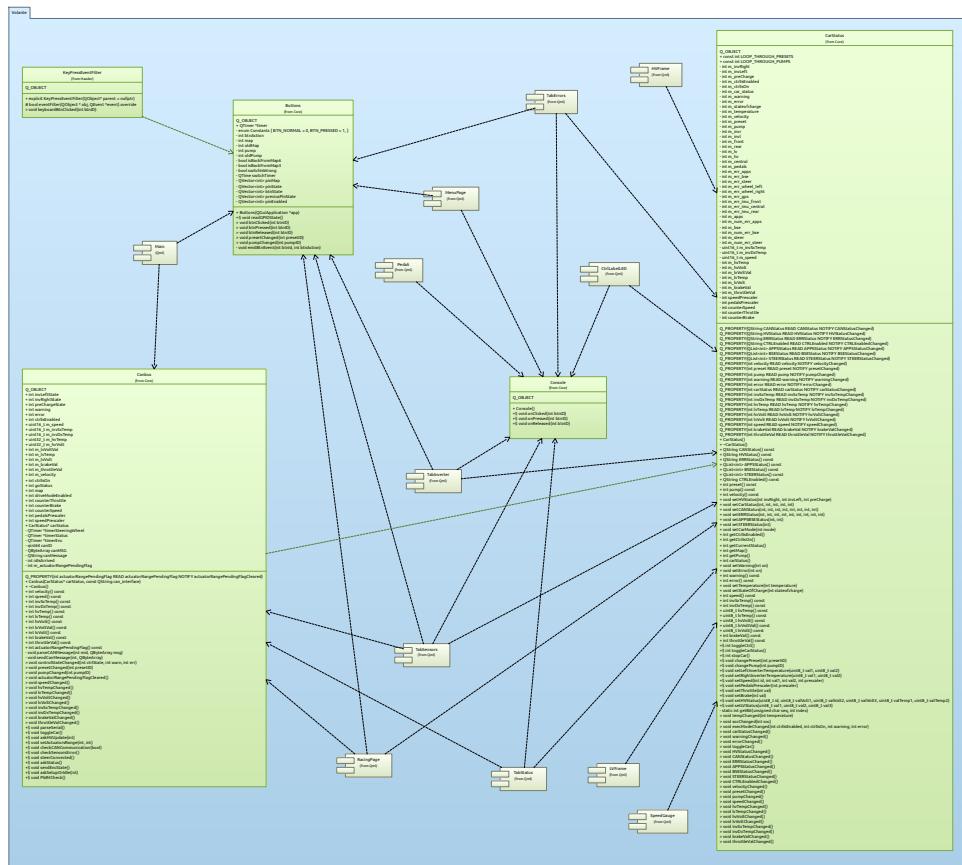
L'accesso ai GPIO è stato gestito utilizzando le librerie di *wiringpi* [3] che, oltre a permetterci di gestire i singoli pulsanti, ci ha permesso grazie al multiplexer di leggere lo stato dei diversi encoder utilizzati come manettini.

Nel Settembre 2018, dopo i risultati ottenuti durante la stagione, ci siamo proposti a *The Qt Company* per avere una sponsorizzazione. Questo ci ha permesso di mettere in relazione l'Università degli Studi di Trento con l'azienda e ampliare la rete dei legami del progetto. Inoltre ha avuto un impatto positivo per il nostro workflow, avere a disposizione le funzionalità complete del framework ci ha permesso di usare l'immagine stock di Qt per Raspberry per la parte di test e la loro *compile-tool* per dispositivi embedded presente nel pacchetto *boot2qt*. Nel pacchetto è presente inoltre una guida basata sui tool di *Yocto*, in particolare la versione *Poky*, per poter costruire da zero una versione base per supportare le applicazioni embedded, per questo ho deciso di abbandonare l'utilizzo di *Buildroot*, in favore di una maggiore documentazione [7].

3 Il Volante

Il confronto con gli altri membri del team ha permesso raccogliere molte idee che hanno dato frutto a scelte tecniche che si sono rivelate nel concreto un buon punto di partenza per ridisegnare la nostra UI e costruire un dispositivo utile in tutte le sue possibili applicazioni. Di conseguenza, per quanto riguarda la parte prettamente grafica e di funzionalità possibili, il confronto è stato essenziale, l'influsso di idee da parte dei partecipanti ha fatto sì che si delineassero dei requisiti essenziali che non erano stati pesanti per la versione di Chimera. Grazie a questo sono poi emersi i limiti dell'utilizzo del Volante mostrandoci cosa poteva essere sviluppato esternamente per completare il quadro del progetto Telemetria, delineando l'assetto del team e gli obiettivi per la stagione 2018/2019.

3.1 Design della Applicazione



le parti di codice non più utili in fase di test, soprattutto per la parte di qml in cui erano presenti pagine di prova e componenti non più utilizzati.

Il programma è diviso in tre parti:

- La lettura dei dati dal Can-Bus (Canbus)
- Lo stato della macchina (CarStatus)
- Il Front-End

La classe *Canbus* si occupa di gestire l'invio e la ricezione dei messaggi via Can-Bus (SocketCan). L'evoluzione, rispetto all'anno precedente, è presente a lato hardware, fornendo alla scheda la possibilità di poter comunicare in Can-Bus direttamente, senza dover passare per una seriale e un dispositivo esterno visto come "gateway". A livello software questo ha permesso di poter utilizzare la libreria di Qt basata sui driver Linux di cui ho già parlato. Questa classe si occupa quindi di inviare messaggi quando richiesto dall'utente o in automatico, di ricevere e settare i valori nella classe *CarStatus*. *CarStatus* si occupa di salvare i valori letti precedentemente dal Can-Bus, in strutture dati in grado di essere lette dall'UI e se ritenuto necessario aggiornare l'interfaccia. Questo non era sempre presente nella versione precedente perché essendo un prodotto in via di sviluppo non sempre questa scelta ci è sembrata la più veloce dovendo aggiornare il codice in tempi brevi. Il vanataggio di Qt è evidente quando si usano le sue librerie, in particolare se consideriamo l'aggiornamento di una digital dash la prima cosa che cambia notevolmente il workflow sono le sue **property** che permettono ai "meta-object" di poter comunicare tra di loro attraverso il sistema *signal* and *slot*. Le property si basano su funzioni per segnalare il cambiamento di un valore (NOTIFY), la sua lettura (READ), la sua scrittura (WRITE) e altre non utilizzate nel nostro caso. Questo permette di rendere accessibile un dato o una struttura dati dal Frontend al Backend e viceversa, in caso di input da parte dell'utente. Il meccanismo inizialmente può sembrare macchinoso, poi, essendo mnemonico, può essere riutilizzato in molte situazioni e, in termini di performance, permette di velocizzare l'accesso ai dati. Come introdotto precedentemente, senza conoscere il framework può sembrare più intuitivo utilizzare delle funzioni in lettura e in scrittura pensate ad hoc, ma questa oltre a non essere la prassi influisce sulle performance e in progetti grandi non permette di mantenere un'architettura precisa. In termini di performance non sono stati fatti test mirati alla raccolta di dati sulla velocità di lettura e scrittura, ma abbiamo visto che alcuni valori non venivano letti soprattutto quando l'eseguibile è sotto stress per i continui messaggi Can-Bus.

3.1.1 Interfaccia 2017

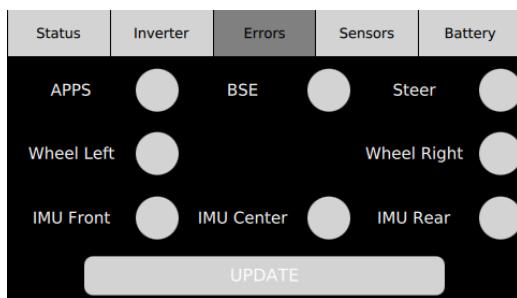


Figura 3.2: Interfaccia di Chimera - Tab View

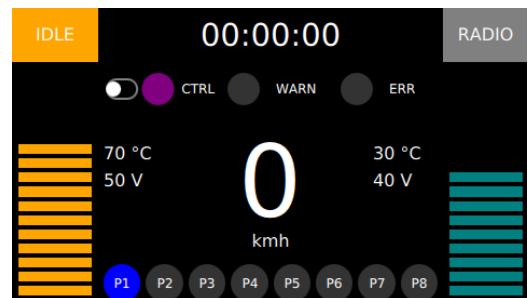


Figura 3.3: Interfaccia di Chimera - Racing Page

La digital dash di Chimera era stata pensata più in un ottica di test, dividendo in due parti le funzionalità. La prima parte, di configurazione e di lettura dello stato della sensoristica in una Tab View. La Tab View necessitava principalmente di due tasti, uno per muoversi ed uscire dalla Tab e un altro per poter entrare e inviare il comando. Considerando il caso della *Tab Errors* con il tasto in alto a sinistra è possibile spostarsi su quella vista, entrare con il tasto in basso a destra e mandare il comando con lo stesso, per poi uscire con il tasto in alto a destra. La seconda parte, la *Racing Page*, dove si poteva vedere lo stato di carica dei pacchi batteria, i Kw erogati, lo stato della macchina e comunicare

via radio con il muretto, funzione che non è mai stata implementata seriamente. Nei prossimi capitoli verrà analizzata in modo più completo l'interfaccia e la *Racing Page*, prendendo in considerazione le scelte fatte per Chimera Evoluzione.

3.1.2 Finite State Machine

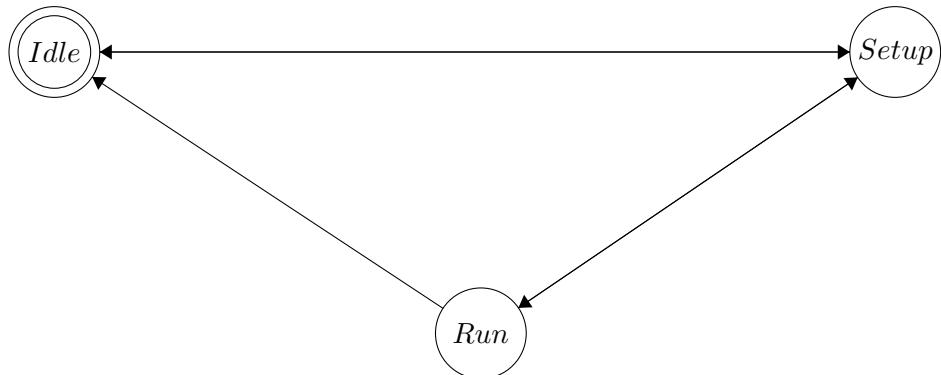


Figura 3.4: Finite State Machine del Volante

Il funzionamento della macchina a stati del Volante è gestita dall'ECU. La FSM rispecchia il lavoro del Team del Controllo, lo scopo del Volante è quello di permettere al pilota di poter cambiare, in sicurezza, lo stato. In sicurezza perchè la maggior parte dei controlli avvengono lato ECU e il Volante è stato sviluppato per rendere il più intuitivo e a prova di errori umani la procedura di accensione. Tramite l'interfaccia possiamo inviare richieste e se sono accettate dall'ECU, quindi se è possibile cambiare stato dallo stato di partenza, effettua i controlli e le richieste a chi di dovere e risponde in modo positivo. La monoposto, successivamente all'accensione del pacco low voltage (BMS LV) si troverà in **IDLE**, da qui se non si presentano errori si può mandare una richiesta di **START** e se la risposta è positiva possiamo quindi passare alla fase di **SETUP**. Durante questa richiesta l'ECU si occuperà di chiedere al pacco high voltage (BMS HW) di accendersi chiudendo in serie il primo e il secondo AIR. Se l'accensione termina con successo il BMS HV risponde all'ECU, la quale poi conferma al Volante lo stato Setup. Il pilota ora ha due possibilità: richiedere l'accensione degli Inverter, e se questa procedura termina correttamente sarà possibile mandare una richiesta per entrare nella fase di **RUN**, in cui l'ECU abilita l'utilizzo della pedaliera. Oppure mandare una richiesta di spegnimento del pacco batteria high voltage attraverso il comando di **STOP**. Il Volante ha due possibilità quando si trova in **RUN**:

- Errori: In base al tipo di errore che riscontra l'ECU può fare richiesta di spegnimento al pacco low voltage o passare in fase di IDLE.
- SetUp: Richiesto dal pilota, disabilita la pedaliera e bisogna ripetere la procedura di accensione degli inverter.

Bisogna sempre tenere in considerazione che le variabili di stato che vengono modificate dipendono dai messaggi presenti nella rete e che hanno come destinario il Volante, dovendo sempre ricevere un feedback da parte dall'ECU L'utente, che sia il pilota o chi sta testando la macchina, ha poche responsabilità in caso di errore. Gli errori che si possono riscontrare sono da verificarsi nei cambiamenti di stato o negli stati stessi della FSM e dalle componenti della monoposto per possibili malfunzionamenti. Solo i manettini, per come è stato scritto il codice, permettono di aggiornare l'interfaccia senza ricevere un feedback in quanto il messaggio sullo stato di connessione del Volante è inviato in ogni 500ms, si per mostrare la presenza nella rete che per condividere le preferenze del pilota.

3.1.3 Cross Compilazione

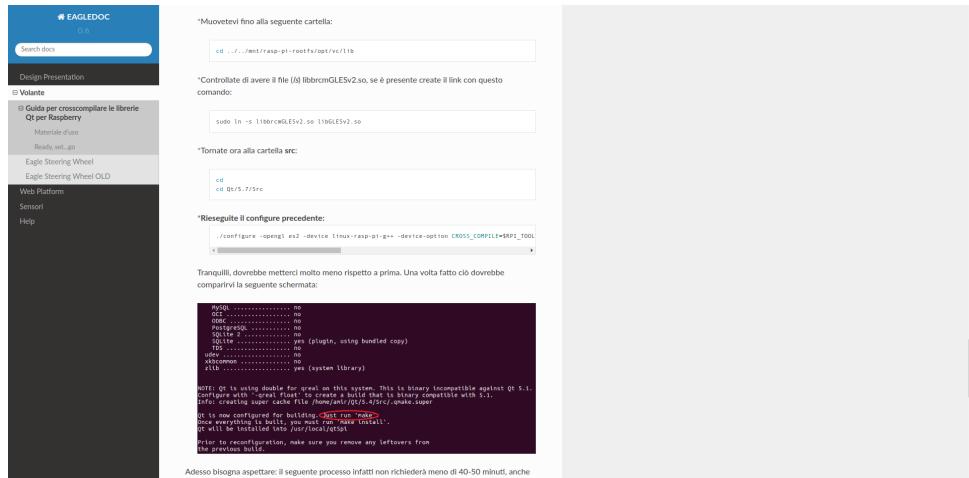


Figura 3.5: Guida Cross Compilazione del Team [4]

L'approccio allo sviluppo del progetto si è diviso in due parti. La prima parte è stata buildare il progetto sul laptop ed eseguirlo, cambiando la classe bottoni per poter avere gli input da tastiera e utilizzare una interfaccia virtuale per la comunicazione Can-Bus. Per l'utilizzo dei bottoni si è deciso di mantenere la posizione de tasti del Volante portandoli sulla tastiera. I quattro tasti funzione sono disposti a "quadrato" sono diventati "Q-A-D-R", per il manettino delle mappe invece sono stati utilizzati i numeri fino al 6. Per quanto riguarda invece gli altri due manettini i test sono stati fatti via ssh valutando l'input e il loro cambiamento di stato, successivamente sono stati portati anche loro su tastiera usando in ordine le lettere della parte destra della tastiera quindi per le pompe "T-Y-U-I-O-P" e per il controllo "F-G-H-J-K-L". La seconda parte ci ha messo più in difficoltà perchè inizialmente abbiamo avuto problemi di compatibilità con il sistema operativo del target device e la sua versione di Qt che non risultava congruente con la nostra. Precedentemente il Volante era stato compilato con la versione 5.7 di Qt, con il nuovo sistema operativo, buildato con Buildroot, potevamo installare solamente la 5.10.1. Questo inizialmente è apparso come un problema da poco ma quando abbiamo visto che per una classe, *QCanBus*, le funzioni non erano più presenti nello stesso modo abbiamo dovuto rivedere la procedura di cross-compilazione per riuscire a compilare il codice con la versione successiva di Qt. Allineate queste divergenze, gli approcci che abbiamo considerato cercando nella documentazione di Qt e nei forum sono due:

- Deploy su Target Board
- Deploy in Locale

Noi abbiamo deciso di compilare il progetto dal nostro host e poi spostare l'eseguibile sul target device, automatizzando la procedura attraverso uno script in bash utilizzando l'auto-login di *ssh*, il kill del processo in esecuzione per poi poter trasferire la versione appena compilata ed eseguirla. La scelta è motivata dal poter lasciare il target device più leggero e non avere necessità di esso nella fase di compilazione, ma solo per eseguirlo. Per compilare il codice del Volante da un dispositivo linux abbiamo dovuto cross-compilare, questo significa che mediante un compilatore esterno abbiamo "emulato" il sistema target compilando il sorgente come se fossimo l'host di riferimento. Così da poter eseguire il file su un architettura diversa da quella del host di origine. Questa è stata la difficoltà più grande riscontrata con l'utilizzo di Qt per dispositivi ARM, abbiamo dovuto trovare il modo di testare in locale l'applicazione, senza avere a disposizione il dispositivo. La procedura per avere l'ambiente adatto a cross-compilare, considerata la versione open-source e non fornendo la tool-chain proprietaria, è molto lenta e richiede ore per poterla portare a termine. La guida che abbiamo ottenuto prevede il montare il sistema operativo del target device (in questo caso non lo stesso, ma raspbian lite) e installare le librerie di Qt, per poi avere il qmake necessario a generare il makefile adatto allo scopo.

La nostra inesperienza nel settore ci ha fatto impiegare molto tempo nella risoluzione di problemi anche banali, ma ci ha anche permesso di imparare tanto e soprattutto di capire che esistono soluzioni più semplici, ancora da testare, delle quali parlerò meglio nelle valutazioni.

3.2 Redesign della Applicazione

L'utilizzo del Framework Qt ha portato numerosi vantaggi quando è stato deciso di ridisegnare l'interfaccia. Inizialmente gli oggetti che componenvano la UI erano tutti nel meta linguaggio *QML* utilizzato da Qt, questi oggetti si basano sul linguaggio JavaScript e permettono di attribuire proprietà all'interfaccia per poi essere gestita dagli eventi generati dal codice C++. Questo in partenza è risultato essere un limite quando si è deciso di utilizzare il formato *.png* per comporre la strumentazione, ma successivamente abbiamo riscontrato la predisposizione di Qt all'utilizzo di immagini.

3.2.1 Dal Low Fidelity Prototype al primo Mockup

Tutto è iniziato da un foglio di carta, un A3 per la precisione, dopo aver raccolto tutti requisti da parte del team abbiamo iniziato a disegnare e a pensare come poteva essere aggiornato il Volante per venire incontro alle nostre necessità.

Le persone coinvolte nel redesign dell'applicazione sono state:

- Davide Farina: Pilota e quindi primo utilizzatore del Volante
- Luca Martinelli: Sviluppatore e utilizzatore del Volante come debugger per la macchina
- Laura Scoccianti Designer, che si è occupata di portare l'interfaccia da Low Fidelity Prototype a Mockup

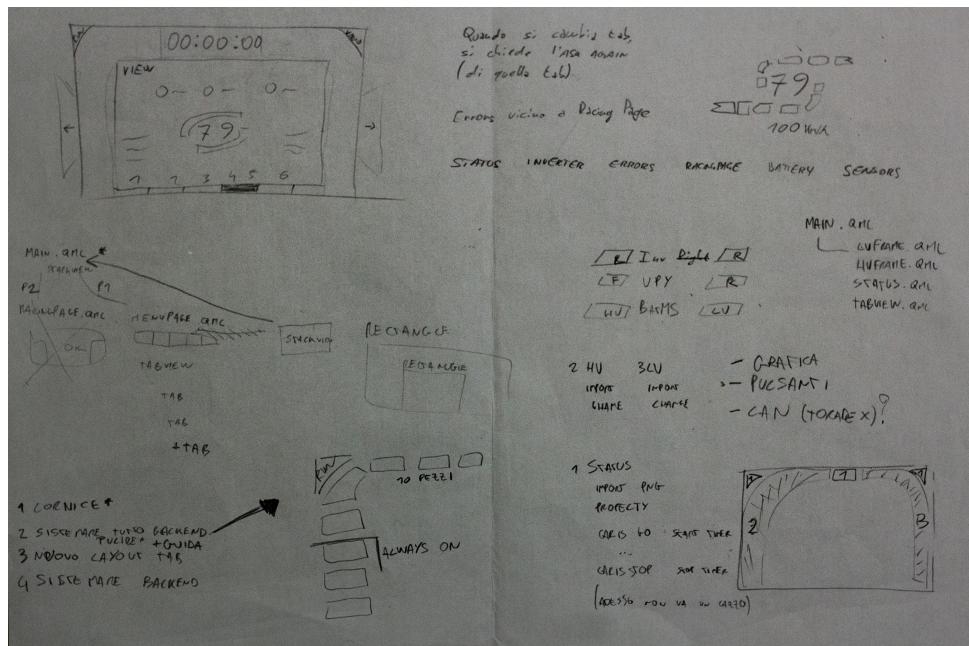


Figura 3.6: Low Fidelity Prototype

Il Low Fidelity Prototype ha preso subito forma, ed una delle cose che è stata in primo luogo fatta presente è la possibilità di potersi muovere in modo intuitivo nelle Tab, quindi senza dover passare da una vista Tab View a una pagina singola come era in precedenza. La giustificazione nasce dall'esigenza di una migliorare usabilità, in quanto anche in fase di *RUN* deve essere possibile cambiare Tab. Questo ha fatto sì che la *Racing Page* dovesse essere integrata nella vista a Tab oppure di cambiare completamente layout. Una delle prime proposte è stata la visuale dello stato della macchina in tutte le pagine, insieme al livello di carica dei due pacchi batteria. Questo ci ha messo davanti ad una scelta, integrare questi indicatori all'interno di ogni pagina o utilizzarli come cornice. La seconda scelta si

è rivelata la più efficace e semplice da implementare, senza dover badare alla riscrittura di codice in più parti. Un'altra funzionalità, che a livello estetico ha permesso di "animare" l'interfaccia in fase di test con il tractive system disattivato, è stata l'inserimento di due slider per i sensori della pedaliera. L'accelleratore e il freno nella Racing Page del Volante sono stati pensati per poter dare un feedback sulla pressione completa, quindi quando si è al 100% durante l'accellerazione. Una proposta che è stata poi scartata è stata la possibilità di visualizzare in una Tab dedicata la dinamica del veicolo potendo valutare la *g-force* letta dalle IMU, l'implementazione si basava su un grafico a radar stile Formula 1.

Per quanto riguarda la scelta di stile e colori da utilizzare per l'interfaccia ci siamo ispirati al mondo della Formula E e dei videogiochi Racing, utilizzando colori molto accesi e facili da distinguere anche in situazioni di scarsa luminosità. Questo è uno dei punti forte della nostra interfaccia, la cura dei particolari e soprattutto la scelta dei colori è stata molto discussa dovendo mostrare e far percepire al pilota in breve tempo il cambiamento di certe condizioni.

Ognuna delle persone coinvolte ha dato la sua opinione sulla fattibilità delle funzioni da integrare e sulla loro utilità votando su quali erano da mantenere e altre da scartare o mettere come ultime nel processo di integrazione. Così facendo si è definito il Gantt per l'integrazione delle funzionalità.

Successivamente è stato necessario portare questo prototipo in qualcosa di più definitivo e integrabile nell'interfaccia. Il risultato è stato reso possibile da Laura Scoccianti, che si è da sempre occupata della parte grafica del team, in questo caso ha portato il nostro Low Fidelity Prototype in un High Fidelity Prototype utilizzando un software per la grafica. Elaborato il primo mockup, dopo averne discusso con i membri del Team siamo passati all'integrazione, ogni singola parte della UI è stata poi esportata in formato .png e integrato con le funzioni già presenti e in corso di sviluppo della macchina.

Questo tipo di approccio è stato reso possibile dalla definizione già in partenza della composizione del team e delle task e basando il processo di sviluppo sui clienti interni ed esterni visti come il "pilota", il "tester" e chi si è occupato della realizzazione.

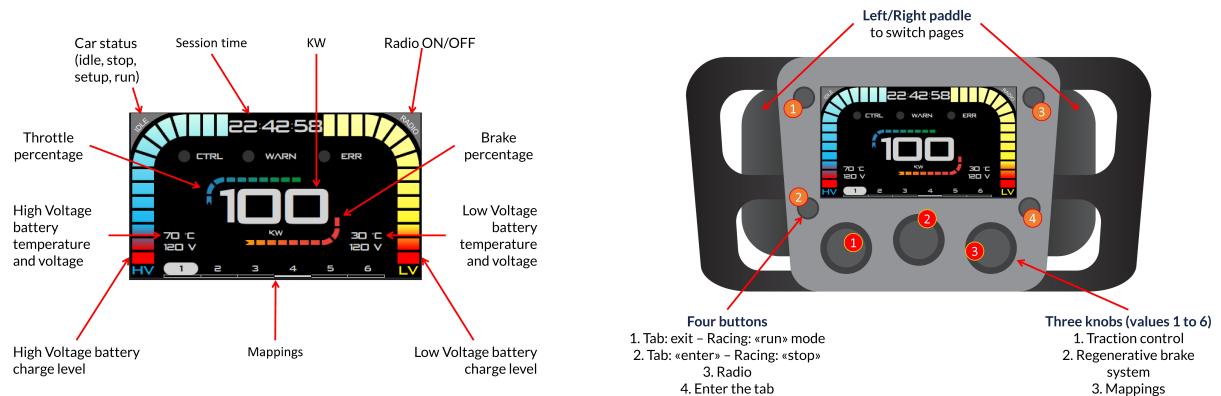
3.2.2 Integrazione della nuova UI del Volante



Figura 3.7: Primo Mockup

Il problema della cornice da integrare è stato risolto ridimensionando la Tab View per fare spazio agli indicatori del pacco HV,LV e inserendo il timer non ancora implementato su Chimera. L'integrazione ha dato problemi nello sviluppo perché la UI non era mai stata pensata per essere responsive, in quanto in un ambito come questo non è effettivamente richiesto. Le difficoltà sono apparse nel dover integrare le

funzioni per gestire il valori da visualizzare inserendoli allo stesso livello nel *main.qml* Tab e corinice. Il ridimensionamento delle Tab ha portato alla perdita della visualizzazione del nome della Tab in cui si è, fattore molto importante perché la mancanza di riferimenti durante l'utilizzo del programma peggiora l'esperienza d'uso. La soluzione è stata indicare la posizione delle Tab nella parte inferiore del display e, grazie l'utilizzo dei paddle, sapere se bisogna andare a destra o a sinistra per visualizzare la Tab desiderata. In più prima non era possibile tornare indietro nelle Tab ma si potevano percorrere solo da sinistra verso destra. Questo fa parte dei compromessi che non si sono rivelati negativi, anzi l'interfaccia di Chimera Evoluzione nasce dalle mancanze riscontrate da Chimera. Una nota sull'interfaccia risulta doverosa, questo non è un dispositivo adatto e utilizzato da tutti, ma solo da chi è stato istruito sul funzionamento della macchina e sulla sua parte elettronica. I nostri piloti sono all'interno del team e lavorano con noi allo sviluppo della monoposto.



L'integrazione dei sensori della pedaliera nella *Racing Page* invece non ha riscontrato particolari problemi, è stata rideimensionata la grandezza del rettangolo in cui dovevano essere collocati i .png che hanno mantenuto le dimensioni del display (il layer in questo caso) quando sono state esportate è stato solo macchinoso farli risultare nella posizione corretta durante l'aggiornamento dei KW. Questo ha anche permesso di rivalutare la grandezza dei font e, verificando direttamente con il pilota, la nostra grafica ci ha fornito i file con le dimensioni aggiornate che si è trattato solo di ri-integrare nella interfaccia. Un'altra integrazione, dovuta all'aggiunta degli encoder sulle ruote, è stato il calcolo della velocità, oltre al valore dei KW richiesti ora è possibile visualizzare in *Racing Page* anche la velocità calcolata dalle schede che si occupano della sensoristica. Il sistema di comunicazione radio, che è stato lentamente abbandonato nel corso dello sviluppo della monoposto, ha lasciato spazio, utilizzando il suo tasto in alto a destra nell'interfaccia, all'invio di un marker in CAN che permette al pilota di segnalare in un certo momento che è successo qualcosa. Durante lo sviluppo si è reso necessario integrare una procedura per calibrare la pedaliera, questo ha messo in risalto alcune criticità nel come era stata pensata la UI mostrando possibili approcci diversi. La calibrazione è stata implementata potendo muoversi nei vari sensori con una procedura interattiva segnalando all'utente se ha effettivamente concluso una delle fasi. Le criticità sono apparse nel momento in cui bisognava mostrare il feedback, cambiando solamente il testo a volte non si permette all'utente di accorgersi se ha effettivamente concluso quella parte. Il problema è stato preso e analizzato e le soluzioni saranno visibili nella prossima implementazione del Volante per la stagione 2019/2020.

3.3 Funzionalità delle diverse pagine

L’interfaccia del Volante di Chimera Evoluzione è composta da più Tab disposte in serie che permettono di seguire in ordine la procedura di accensione e di test standard della monoposto. Alcune funzionalità sono state inserite solo a livello grafico e per dare forma all’interfaccia, sono state successivamente implementate su Chimera Evoluzione. I nomi delle Tab hanno subito modifiche perché per come era stata concepita non erano presenti vere relazioni tra il nome, le funzionalità e ciò che doveva essere visualizzato. La versione mostrata è quella standard utilizzata per i test.

3.3.1 Warning Tab



Figura 3.8: Chimera Evoluzione - Tab Warnings

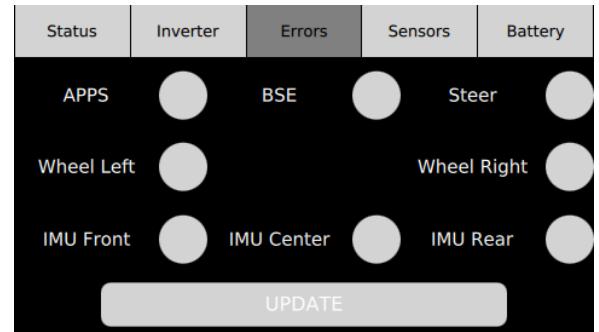


Figura 3.9: Chimera - Tab Errors

La prima Tab, chiamata anche *Warning Tab*, mostra lo stato della sensoristica. Diversamente dalla *Errors Tab*, i warnings non sono errori critici e servono solo a informare il pilota se certi sensori non funzioneranno come dovrebbero. Inizialmente erano presenti alcuni sensori che sono stati rimossi perché non più necessari, e la richiesta di update è stata resa automatica per poter aumentare lo spazio nella Tab. Nelle prime fasi di test questa funzionalità si è rivelata molto utile perché ci ha permesso di vedere sin dalla prima accensione se i dispositivi erano collegati e funzionavano. Con la presenza di warnings è possibile abilitare il tractive system e utilizzare la macchina, se dovessero apparire degli warnings durante la fase di *Run* questo non cambierebbe lo stato, cosa che invece accade per gli errori.

3.3.2 Error Tab

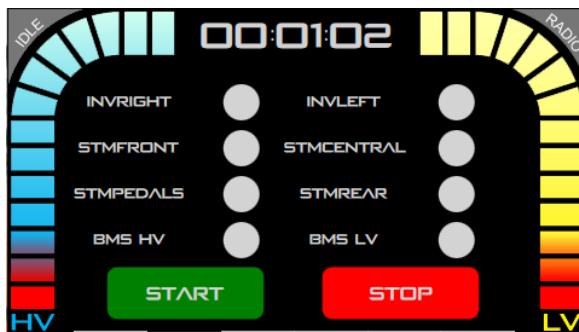


Figura 3.10: Chimera Evoluzione - Tab Errors

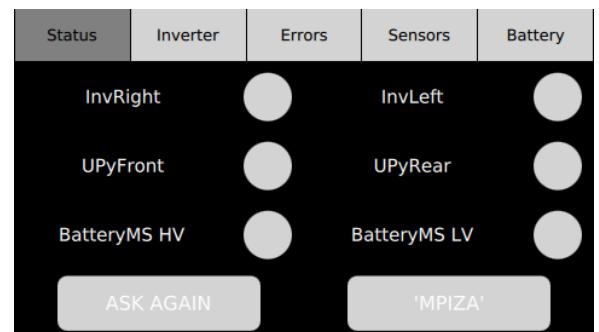


Figura 3.11: Chimera - Tab Status

La seconda Tab, chiamata anche *Errors Tab*, permette di visualizzare la panoramica degli errori presenti in macchina, se sono presenti errori critici andranno verificati con gli interessati e risolti per poter entrare nella fase di *Setup*. In questa Tab è stata ripensata la procedura di accensione, cambiando il comando di *Ask Again* con il comando di *Stop* ed è stata ridimensionata per poter inserire altre schede come ad esempio la pedaliera. Se non sono presenti errori, possiamo mandare il messaggio di *Start* e successivamente poter tornare nella fase *Idle* tramite il comando *Stop*.

3.3.3 Status Tab

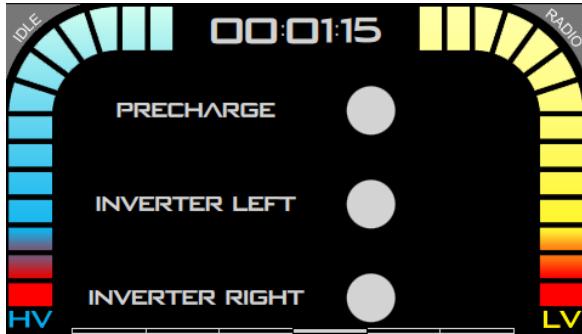


Figura 3.12: Chimera Evoluzione - Tab Status

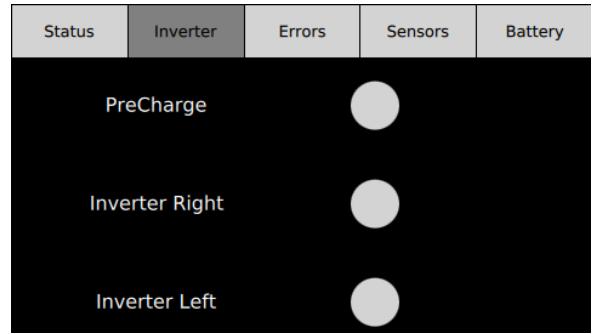


Figura 3.13: Chimera - Tab Inverter

La terza Tab, chiamata anche *Tab Status*, mostra lo stato della precharge e degli inverter. Da qui, grazie ai led, possiamo vedere se non comunicano, quindi led grigio, se sono abilitati o no (rosso e verde). Questa Tab permette di accendere o spegnere gli inverter facendone richiesta. Questa procedura deve essere effettuata prima di entrare nella *Racing Page* per poter entrare nella modalità *Ready To Drive*. Non è possibile entrare in questa modalità, se non vengono abilitati, in quanto di default sono impostati ad off.

3.3.4 Racing Page

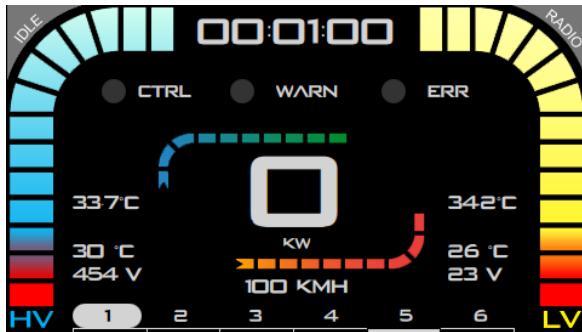


Figura 3.14: Chimera Evoluzione - Racing Page

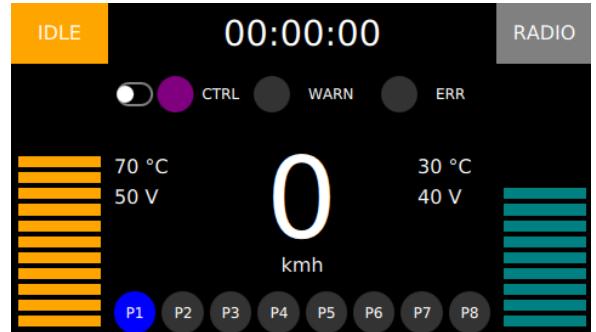


Figura 3.15: Chimera - Racing Page

La quarta Tab, chiamata anche *Racing Page* è la più importante e la più curata dal punto di vista del design. Senza doversi spostare in altre Tab si è in grado di avere una visione generale della macchina. Da qui si può interagire con la sezione in alto a sinistra che mostra lo stato della macchina (idle, setup, run, stop) attraverso il comando di *Run* e il relativo comando di *Stop* che nel primo caso abilitano l'utilizzo della pedaliera e nel secondo permettono di tornare nella fase di *Setup*, disabilitando gli inverter. Nella parte più alta sono presenti tre led che mostrano se il controllo è stato abilitato e in sequenza i possibili warning o errors presenti in macchina in caso di malfunzionamenti durante la fase di *Run*. L'introduzione di questi led è stato inoltre utile per evitare che il pilota possa entrare, se non abilitato, nella fase di *Run* tenendo sempre a mente che il Volante non si occupa di gestire i casi di errore, ma solo di visualizzarli. Le funzioni disponibili, come già spiegato, sono definite dalla macchina a stati. Al centro dello schermo è posizionato il valore, in KW, richiesto ai motori, attorno possiamo trovare in azzurro quanto viene premuta la pedaliera e in rosso lo stato del freno per dare un feedback al pilota mostrandogli, in accelerazione, quando si trova al 100%. Appena sotto invece è posizionata la velocità letta dagli encoder, fino ad arrivare alla mappa scelta. In questa sezione il pilota può vedere in che mappa si trova, scelta grazie all'ausilio dei manettini. Le mappe permettono di selezionare, in %, la potenza da erogare ai motori, la prima mappa abilita la retromarcia (-20%), dalla seconda in poi incrementa del 20% fino ad arrivare alla mappa 6 che indica il massimo erogabile. Il selettore delle mappe oltre ad essere molto utile in fase di test, soprattutto per verificare in linea generale la sicurezza della macchina apre molte possibilità in termini di personalizzazione della guida. Il pilota infatti oltre a questo manettino ne ha a disposizione altri due: uno per impostare il

traction control e l'altro per il sistema di cooling. Grazie al primo il pilota può decidere se abilitare il torque vecoring, disattivarlo e utilizzare o no lo slip control e con l'ausilio del secondo può impostare, manualmente, l'utilizzo delle pompe che indipendentemente da questo funzionano autonomamente. Rispettivamente sulla parte sinistra e destra troviamo lo stato del pacco low voltage e high voltage con le loro temperature e voltaggio. In più, durante la fase di test, si è reso necessario mostrare la temperatura degli inverter, da cui si è deciso di inserire questi dati congruentemente alla posizione degli inverter in macchina, quindi sinistra per l'inverter di sinistra e destra per l'inverter di destra.

3.3.5 Battery Tab



Figura 3.16: Chimera Evoluzione - Tab Battery

Status	Inverter	Errors	Sensors	Battery
HV	LV			
T. Max:0 ° C	V1:0 V			
T. Avg:0 ° C	V2:0 V			
T. Min:0 ° C	V3:0 V			
V. Max:0 V	V4:0 V			
V. Avg:0 V	V5:0 V			
V. Min:0 V	Tot. E:0 J			

Figura 3.17: Chimera - Tab Battery

La quinta Tab, chiamata anche *Battery Tab*, permette di visualizzare lo stato di carica del pacco batterie High Voltage e Low Voltage. Per entrambe è presente la temperatura minima, massima e media della temperatura e della carica dei due pacchi. Questa Tab è da utilizzare in fase di test, in quanto al pilota non avrà, o solo in pochi casi particolari, la necessità di sapere lo stato delle celle. Le informazioni più generali sono presenti invece nella *Racing Page*.

3.3.6 Sensor Tab

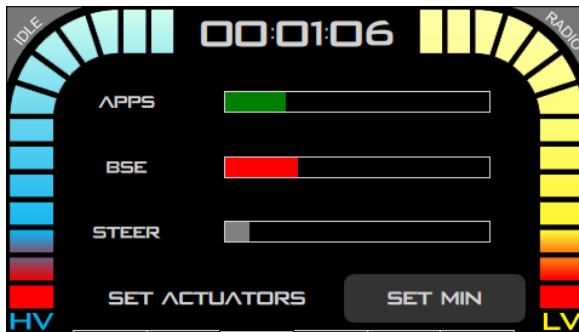


Figura 3.18: Chimera Evoluzione - Tab Sensors

Status	Inverter	Errors	Sensors	Battery
APPS			● ● ● ● ●	
BSE			● ● ● ●	
STEER			●	
Set Actuators [MIN,MAX]				SET

Figura 3.19: Chimera - Tab Sensors

La sesta Tab, chiamata anche *Tab Sensori*, è composta dai valori raccolti dalla pedaliera e dallo sterzo (APPS, BSE and STEER). In questa Tab è possibile vedere attraverso una barra orizzontale che valori assumono, in modo indicativo, i sensori di riferimento. La pedaliera è composta dall'acceleratore e dal freno, che come già spiegato, in una scala da 0 a 10 sono visualizzabili anche nella *Racing Page*. Per tutti e tre i sensori si è reso necessario l'implementazione di una procedura di calibrazione, nella quale il pilota o chi testa la sensoristica è in grado di far memorizzare il valore massimo e minimo alle schede di riferimento. Questa procedura ci è stata molto utile durante i test ed essendo interattiva è anche facile da seguire.

3.4 Considerazioni sullo Sviluppo dell'Interfaccia

Il lavoro che è stato fatto ha messo al centro dello sviluppo il cliente, considerato nel nostro caso il pilota e i suoi utilizzatori che lo vedono come strumento di debug, che è stato da sempre membro attivo durante lo sviluppo. La metodologia utilizzata ha permesso di migliorare l'usabilità mostrando un'attenzione particolare per il punto estetico e rafforzare la flessibilità del prodotto, che considerando la nostra competizione, non permette di ridisegnare da zero una componente come questa in soli 8 mesi di tempo. L'aggiornabilità del prodotto sia nel hardware che nel software ci obbliga a riflettere molto sulle scelte che prendiamo evitando di prendere strade che possono risultare troppo impegnative, trovando ostacoli, come ad esempio il prezzo e il tempo per imparare ad utilizzare nuove tecnologie.

La scelta di mettere al centro dello sviluppo il cliente ha mostrato diverse fasi che possono essere raggruppate e schematizzate che si sono rivelate necessarie alla conclusione del progetto.

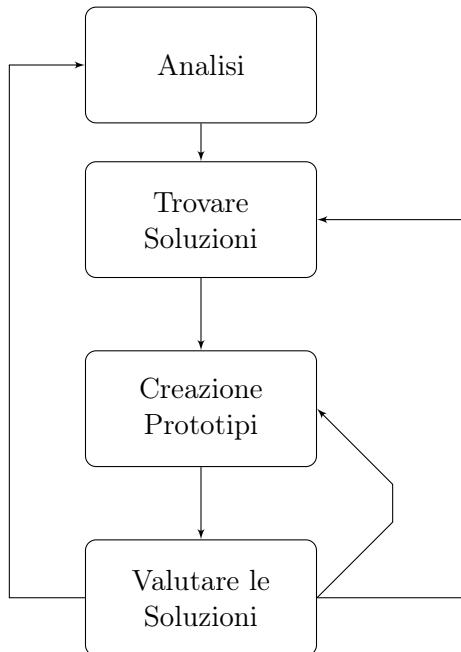


Figura 3.20: User Centered Design Process

- Analisi: Identificare i bisogni, il contesto e stabilire i requisiti
- Design: Generare tutte le possibili soluzioni
- Sviluppo: Creare prototipi da poter mostrare al cliente
- Valutazioni: Valutare il prototipo, Identificare i pro e contro.

Questo è stato uno dei motivi che ci ha permesso di sviluppare evitando di correre rischi trovando sempre i compromessi più adatti senza mai perdere di vista l'obiettivo finale. Fornire una soluzione vincente alla squadra.

4 Valutazioni

Attraverso l'implementazione delle nuove funzionalità, il miglioramento dell'usabilità grazie allo sviluppo della nuova interfaccia è stato possibile avere uno strumento di debug per leggere i messaggi in Can-Bus, salvarli grazie agli applicativi di *canutils* e ottenere quindi uno software stabile per testare e fare operazioni di debug sulla macchina. Il Volante di Chimera Evoluzione è pienamente utilizzato in tutte le sue funzionalità ed è ritenuto lo strumento sviluppato dal team più stabile in macchina al momento. Questo oltre alle scelte tecniche prese che lo rendono performante in diverse situazioni, mostra i vantaggi nella sua architettura, cioè essere principalmente in lettura.



Figura 4.1: Presentazione del Volante - Varano Luglio 2018

Nel Luglio 2018, durante la partecipazione alla tappa italiana del campionato presso il circuito di Varano (PR), ho avuto la prima conferma da parte di esterni sulla qualità del nostro lavoro, delle scelte tecniche e di sviluppo che sono state fatte durante l'anno. La riconferma di ciò è stata la premiazione da parte di Alessandro Pocecco, Project Leader della parte Elettronica di Lamborghini, con il premio per *Best HMI Solution*. La soluzione che abbiamo deciso di presentare è stata valutata non solo come prodotto, ma soprattutto per la presentazione che ho avuto l'onore di condurre, mostrando in particolare il nostro approccio durante la fase di design del prodotto e le metodologie di sviluppo. Il giudice si è complimentato per l'impegno, dandoci numerosi consigli e facendo apprezzamenti sulle scelte tecniche in base ai nostri requisiti, che sono state motivate utilizzando un approccio molto più vicino al mondo lavorativo che a quello didattico.

Ad Agosto 2018, durante la tappa di Barcellona, ci è stato fatto presente della mancanza di una vera soluzione telemetrica, questo perchè il Volante è stato pensato per essere utilizzato principalmente dal Pilota e per i test, ma non per la raccolta dati. Da qui è nata la necessità di sviluppare una applicazione a microservizi basata sulla comunicazione wireless della macchina con un dispositivo esterno per raccoglierne i dati e visualizzarli in real-time con il salvataggio in database per poter poi fare delle analisi più precise sulla nostra monoposto.

4.1 Sviluppi Futuri

Per quanto riguarda il Volante, grazie ai feedback dei giudici, delle persone che ci hanno lavorato come utenti e da chi l'ha sviluppato abbiamo raccolto molte informazioni sul come poterlo migliorare. Per poter avere un'analisi più precisa sulle risorsse da impiegare al miglioramento del dispositivo abbiamo deciso di utilizzare un'approccio qualitativo diverso. L'approccio consiste nel definire parametri misurabili in fase di progettazione, molto simile a quello utilizzato dal gruppo telemetria ma senza l'utilizzo di variabili. Questa tecnica viene chiamata **QFD** e consiste nel tenere conto delle necessità del cliente esterno per poter venire il più possibile incontro a quelle del cliente interno (sviluppatori e designer). È importante precisare che il nostro principale competitor siamo noi stessi, con la soluzione che abbiamo implementato nel 2018. Al momento non sono presenti all'interno della competizione di Formula SAE prodotti custom simili al nostro, curati sia dal punto di vista grafico che funzionale, ma soluzioni per il motor-sport in generale acquistate e utilizzate dalle squadre.

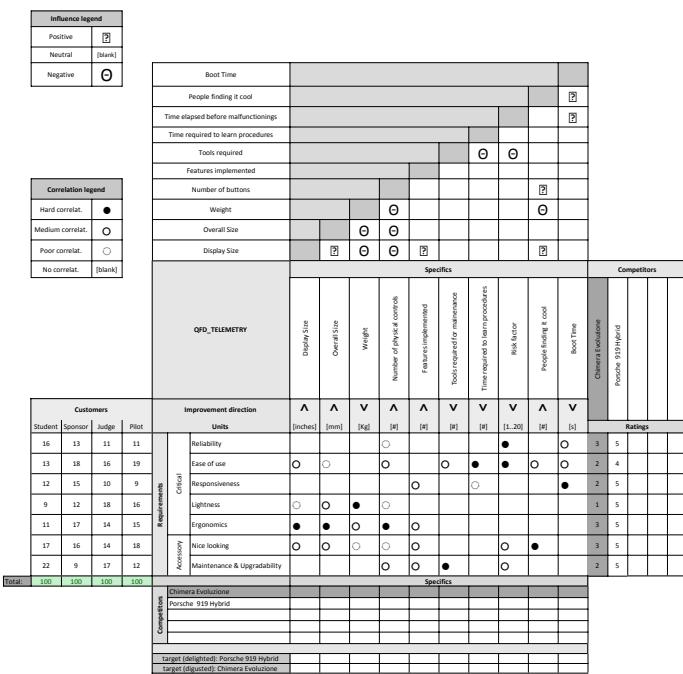


Figura 4.2: QFD Ottobre 2018

Dalla QFD sono emersi diversi punti importanti su cui lavorare, qui schematizzati:

- Case ridisegnato per renderlo facilmente aggiornabile grazie al ausilio di una mascherina che può essere rimossa per poter lasciare il case uguale e ridurre i costi.
- Impugnatura ergonomica su misura ottenuta attraverso scanner 3D.
- Cambiare hardware, scelta di una scheda più potente e ridurre lo spazio della componentistica hardware, la soluzione di riferimento è un Raspberry Pi CM3 con memoria SD.
- Gestire su due Thread il programma, uno per la lettura del Can-Bus l'altro per l'interfaccia.
- Migliorare i tempi di boot del sistema operativo. Grazie al premio vinto abbiamo potuto contattare *Qt* per una sponsorizzazione e ora abbiamo a disposizione l'intera suite per lo sviluppo embedded che, attraverso anche alla collaborazione dei loro partner, ci permetterà di utilizzare al massimo le potenzialità del loro framework.

- Aggiornare l’interfaccia proponendo più versioni in base al caso d’uso e il pilota.
- Inserire led di notifica sopra al display, questa funzionalità è ancora da definire ma una sua prima applicazione può essere notificare certe informazioni al pilota o a chi fa i test e in fase di gara vedere il bloccaggio delle ruote.
- Aggiornare i dispositivi di input, inserendo altri due paddlee un bottone, uno per il launch control e l’altro da utilizzare come marker (ripristinando la soluzione della comunicazione radio).
- Aumentare la dimensione del display da 4.3" a 5", per poter integrare nuove funzionalità e avere un display qualitativamente migliore.
- Rendere ancora più intuitiva l’interfaccia grazie a tasti con scritte colorate e automatizzare alcune procedure alle quali il pilota non interessa conoscere attraverso un bottone di "start automatico", mantenendo e migliorando la vecchia procedura.
- Aggiungere la possibilità di premere più tasti contemporaneamente per attivare altre funzioni.
- Migliorare il feedback per il pilota, attraverso il flash di colori e informazioni, così da poter essere informati più velocemente sullo stato della macchina.



Figura 4.3: Primo Mockup del Volante di Fenice

Il progetto del Volante non è da intendersi come finito, ma come continua ricerca di soluzioni migliori. Il test su altri dispositivi e con altre tecnologie mostra la facilità/difficoltà di integrarsi al nostro progetto, la scelta di effettuare questo tipo di prove viene preso dal singolo valutando la fattibilità e l’interesse personale. Un esempio è il framework che utilizza il gruppo Volkswagen, in particolare Audi con il loro premiato MMI. È sicuramente un ottima soluzione, ma le considerazioni fatte sui prezzi delle licenze, il target dei loro clienti (principalmente designer) ci ha spinto scartarlo optando per soluzioni open-source, poco costose e che sfruttano tecnologie che possono mostrarsi polifunzionali.

5 Conclusioni

L'esperienza all'interno del team mi ha permesso di applicare le conoscenze informatiche e maturarne altre in settori che non avrei mai potuto conoscere se non grazie ad un progetto del genere. I sistemi embedded e il design dei prodotti mi è sempre piaciuto ed avere la possibilità di lavorarci è stato motivo di grande soddisfazione oltre che avermi permesso di individuare i possibili sviluppi di un percorso professionale futuro. Lavorare sul volante e collaborare con altre persone, professori e studenti, mi ha aiutato a fare mie metodologie di sviluppo e le tecnologie utilizzate. Da Settembre 2018 mi sono occupato personalmente della parte software del Volante, grazie alle conoscenze maturate nel campo dei sistemi operativi e dello sviluppo con Qt, lasciando la parte elettronica e meccanica a Daniele Faccinelli e Ciro Malacarne. L'esperienza complessiva mi ha permesso di entrare in contatto con persone altamente specializzate nel settore dell'automotive, che riconoscono i nostri sacrifici e il lavoro che c'è dietro ad un progetto di questo tipo. Reputo il progetto molto impegnativo, ma grazie all'avanzamento dei lavori del Volante e l'entusiasmo della squadra e dei collaboratori, ho intenzione di continuare l'esperienza nel team per terminare il progetto telemetria e lo sviluppo del Volante.

Bibliografia

- [1] Repository del volante di chimera e chimera evoluzione. <https://github.com/eagletrt/chimera-telemetria>.
- [2] Repository del volante di fenice. <https://github.com/eagletrt/fenice-steeringwheel>.
- [3] wiringpi. <http://wiringpi.com/extensions/i2c-mcp23008-mcp23017/>.
- [4] Antonio Stefani e Luca Martinelli. Cross compilazione per il volante. <https://eagletrt.github.io/volanteCrosscompilazione.html>. ultimo accesso 17/06/2019.
- [5] Amir Mansouri. How to cross compile qt for raspberry pi 3 on linux (ubuntu) for beginners! <https://medium.com/@amirmann/how-to-cross-compile-qt-for-raspberry-pi-3-on-linux-ubuntu-for-beginners-75acf2a078c>. ultimo accesso 01/06/2019.
- [6] Luca Martinelli. E-agle trt documentazione. <https://www.eagletrt.github.io>. ultimo accesso 17/06/2019.
- [7] Scott. Build di un sistema operativo con qt per raspberry pi. <https://jumpnowtek.com/rpi/Raspberry-Pi-Systems-with-Yocto.html>.