POLITECNICO DI TORINO

I Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

Relazione finale di tirocinio

Sviluppo di un'interfaccia uomo-macchina per On-Board Unit

Gabriele Degola

Tutori accademici:
prof. Gianpiero Cabodi
prof. Massimo Violante

Tutore aziendale:
ing. Daniele Brevi

Azienda ospitante:
Fondazione LINKS
Via Pier Carlo Boggio 61, Torino (TO)

A.A. 2018/2019

Indice

1. Introduzione

Il tirocinio oggetto di questa relazione, svolto dal sottoscritto Gabriele Degola da marzo a giugno 2019 presso la Fondazione LINKS per un totale di 250 ore, ha come obiettivo lo sviluppo di un'applicazione che, basandosi sui dati raccolti da una On-Board Unit sviluppata dalla stessa azienda, li visualizzi in tempo reale e in maniera fruibile per gli utenti su PC, smartphone o tablet.

La fase di sviluppo è stata preceduta da una fase di analisi delle richieste e delle tecnologie già utilizzate dall'azienda per individuare la migliore soluzione dal punto di vista software, in particolare per scegliere se realizzare l'interfaccia come applicazione nativa o come Progressive Web App. Si è trattato infatti di un progetto completamente nuovo e che ho seguito fin dall'inizio.

Dovendo elaborare in tempo reale i dati che vengono ricevuti dalla On-Board Unit, è stato necessario implementare un server per trasmetterli immediatamente all'applicazione tramite una connessione WebSocket, modificando alcune componenti già sviluppate dall'azienda.

Le diverse fasi del tirocinio possono quindi essere così riassunte:

- analisi del contesto di lavoro;
- progettazione e realizzazione di un server necessario per trasmettere i dati dalla OBU all'applicazione;
- progettazione e realizzazione dell'interfaccia grafica e dell'applicazione.

Il risultato è un'applicazione web completa in grado di elaborare graficamente le informazioni ricevute e mostrarle in tempo reale all'utente tramite diverse modalità di visualizzazione. Durante lo sviluppo il codice è stato commentato accuratamente per permettere la generazione automatica di documentazione con strumenti opportuni. Per lo sviluppo dell'interfaccia si è lavorato su una macchina virtuale con sistema operativo Ubuntu 18.04.

2. Contesto

La Fondazione LINKS – Leading Innovation and Knowledge for Society è un'istituzione senza scopo di lucro fondata nel 2017 dalla Compagnia di San Paolo e dal Politecnico di Torino, in cui sono confluiti l'Istituto Superiore Mario Boella (ISMB), centro di ricerca applicata nel campo delle ICT, e l'Istituto Superiore sui Sistemi Territoriali per l'Innovazione (SiTI), che si occupa di ricerca e formazione orientate all'innovazione e alla crescita socio-economica dalla logistica alla sicurezza del territorio. La missione della Fondazione è rafforzare l'interazione fra ricerca e mondo delle imprese e finalizzarla all'internazionalizzazione del sistema socioeconomico locale.

In particolare, l'Area di Ricerca Multi-Layer Wireless Solutions dell'ex ISMB studia soluzioni di accesso, architetture di rete e applicazioni per l'erogazione di nuovi servizi su canale mobile, oltre a occuparsi di comunicazione veicolare. Il tirocinio è stato svolto in questa area dell'azienda, sotto la supervisione degli ingegneri Daniele Brevi, Guido Alejandro Gavilanes Castillo e Edoardo Bonetto. Il responsabile dell'area di ricerca è il dottor Riccardo Scopigno.

La Fondazione LINKS è partner del progetto AUTOPILOT, finanziato dall'Unione Europea per circa 20 milioni di euro con l'obiettivo di sviluppare un'infrastruttura di veicoli a guida autonoma in un ambiente connesso basato sull'Internet of Things. Il progetto fa parte del più ampio IoT European Large-Scale Pilots Programme, il quale comprende diversi progetti per promuovere lo sviluppo e l'evoluzione dell'Internet of Things in Europa. AUTOPILOT segue lo sviluppo e il test di veicoli a guida autonoma in cinque modalità di guida:

- Urban Driving, per il test di veicoli completamente autonomi in situazioni complesse;
- Automated Valet Parking, per il controllo dei veicoli in aree di parcheggio;
- Platooning, per il controllo di veicoli in coda;
- Real Time Car Sharing, per l'implementazione di veicoli a guida autonoma in servizi di car sharing;
- Highway Pilot, per la gestione di eventi stradali tramite una rete di sensori.

Lo sviluppo e i test sono seguiti da 45 aziende partner in sei Pilot Site, a Tampere (Finlandia), Versailles (Francia), Livorno (Italia), Daejeon (Corea del Sud), Brainport (Paesi Bassi) e Vigo (Spagna).

La Fondazione LINKS opera nell'area di Livorno, dove sono studiati gli use cases Urban Driving e Highway Pilot. Il sito di Livorno e i due use cases sono quindi discussi più dettagliatamente.

2.1. Il Pilot Site di Livorno

L'area italiana per i test del progetto AUTOPILOT si trova tra Firenze e Livorno e comprende un tratto della strada di grande comunicazione Firenze-Pisa-Livorno e un'area interna

al porto di Livorno. In quest'area sono coinvolte diverse aziende o enti, tra cui il CNIT (Consorzio Nazionale Interuniversitario per le Telecomunicazioni), AVR, Continental, il Centro Ricerche Fiat, Thales Italia, TIM e appunto la Fondazione LINKS.

Qui i dispositivi IoT sono posizionati a lato della strada e a bordo delle sette auto utilizzate, due connesse e con possibilità di guida autonoma e cinque solo connesse. Nei test in ambito urbano viene utilizzato anche un prototipo di bicicletta connessa. Le aziende partner gestiscono il controllo, la gestione e la comunicazione tra i dispositivi.

2.2. Highway Pilot

Questo use case prevede test su auto a guida autonoma supportate dall'Internet of Things, in movimento su una strada connessa. Le auto sono equipaggiate con dispositivi che eseguono autonomamente scelte di guida, come la modifica della velocità o il cambio di corsia, attivati dall'IoT. Sulla strada si trova un sistema di comunicazione basato su una rete di sensori che raccolgono informazioni e le rendono disponibili alle applicazioni basate sul cloud, con l'obiettivo di ridurre il rischio di incidenti in caso di imprevisti sulla strada, come lavori stradali o allagamenti. Per ragioni di sicurezza le auto a guida autonoma sono seguite da veicoli connessi a guida umana e le informazioni ricevute dai sensori sono controllate e ritrasmesse da un centro di controllo.

In presenza di un allagamento, dei sensori a bordo strada rilevano la presenza di acqua sulla strada e inviano dei messaggi, i quali sono ricevuti dal veicolo a guida autonoma con un certo anticipo. In questo modo il veicolo ha la possibilità di frenare dolcemente e raggiungere il tratto allagato con velocità adeguata.

Nel caso di lavori stradali, il tratto di strada interessato è segnalato dagli operatori umani con informazioni su limiti di velocità e corsie chiuse. In questo modo il veicolo a guida autonoma può ridurre la velocità avvicinandosi ai lavori, cambiare corsia, procedere alla nuova velocità consentita e aumentare nuovamente la velocità quando l'evento è superato.

2.3. Urban Driving

Questo use case prevede test su un ambiente simil-urbano all'interno del porto, con auto a guida autonoma, auto connesse, pedoni in corrispondenza di attraversamenti pedonali e biciclette connesse, per studiare l'effetto dell'Internet of Things sulla sicurezza di pedoni e ciclisti. È ovviamente presente un centro di controllo.

La grande presenza di sensori in un ambiente simile permette ai veicoli connessi di ottenere una descrizione accurata dell'ambiente circostante, in anticipo rispetto ai veicoli tradizionali, con la possibilità di modificare preventivamente il proprio comportamento per evitare scontri e aumentare la sicurezza. Ad esempio, in un ambiente connesso la caduta di un ciclista sarebbe immediatamente segnalata dai dispositivi IoT, oppure un semaforo connesso potrebbe rilevare se

un pedone sta attraversando la strada con luce rossa. In questo modo i veicoli a guida autonoma in avvicinamento conoscerebbero in anticipo la posizione dell'evento, potendo regolare la velocità o fermarsi di conseguenza.

2.4. Contesto lavorativo

Per il progetto AUTOPILOT l'area di ricerca sta sviluppando un modello di On-Board Unit (OBU), connessa ai dispositivi IoT tramite una connessione wireless per ricevere messaggi e inviarne a sua volta.

La OBU sviluppata dalla Multi-Layer Wireless Solution Area

Per visualizzare i messaggi in una forma comprensibile, gli ingegneri Guido Gavilanes e Francesco Scappatura hanno scritto un programma in linguaggio C, chiamato hmimon, per interrogare continuamente il server interno della OBU e stampare a video i dati ricevuti. A seconda dei parametri passati su linea di comando vengono mostrati dei contatori aggiornati a ogni messaggio, divisi per tipo di evento e con una sintassi standard del settore automotive, oppure le vere e proprie risposte del server in forma di messaggi JSON. Al server vengono inviate query PostrgreSQL che possono essere aggiornate modificando il codice sorgente, per seguire le frequenti modifiche alla OBU o ai sensori e le nuove funzionalità aggiunte.

L'hmimon riceve e stampa a video i seguenti messaggi:

- POTI: informazioni sull'egovehicle, cioè il veicolo su cui è posizionata la OBU. Contiene il tempo attuale, le coordinate geografiche e informazioni sulla velocità e direzione del veicolo.

POTI {"UTC" : 1554819694939, "dLat" : 436202750, "dLon" : 104486820, "spdsens" : 0, "spd" : 1892, "head" : 2295, "yawrate" : 0, "curval" : -2586, "curvperc" : 100}

- ROC: informazioni sugli altri veicoli connessi. Contiene un identificativo del veicolo, il tempo attuale, la posizione relativa rispetto all'egovehicle (calcolata con un complesso algoritmo), la probabilità che questa sia corretta e le coordinate geografiche del veicolo. Inizialmente, per una svista nel codice il JSON conteneva due volte la chiave "lat" e non era perciò possibile interpretarlo correttamente.

ROC {"v_id" : 3906, "ts" : 1554819698.19495, "class" : "BehindFarLeft", "prob" : 1.0000, "lat" : 436204890, "lat" : 104490710}

- PHTAB e SCOR: informazioni di debug, i PHTAB contengono informazioni sulla storia dei veicoli e indicano la loro posizione relativa rispetto all'egovehicle.

PHTAB {"id" : 3906, "age" : 2, "ph_x" : 345.649, "ph_y" : 965.363, "ph_trx" : -943.164, "ph_try" : -402.295, "ph_lat" : 436229380, "ph_lon" : 104599790, "time" : 1554819688709}

SCOR {"ts" : 1554818395807, "v_id" : 3906, "onc" : 0.0000, "ong" : 1.0000, "beh" : 0.0000, "ahe" : 0.5159, "rig" : 0.0000, "lef" : 1.0000, "pimp" : 0.0000, "far_rig" : 0.0000, "far_lef" : 1.0000}

- HMI: informazioni sugli eventi stradali, i quali condividono la stessa descrizione ma contengono informazioni differenti:
    - i messaggi relativi a imprevisti stradali come allagamenti o lavori in corso indicano il tipo di evento, il nuovo limite di velocità se presente, la distanza dall'egovehicle dell'evento e dell'inizio del nuovo limite e la sua priorità;

    HMI {"ROAD_SplD" : 141, "Trul" : 2, "EvtD" : 299, "App" : 1, "HPri" : 1, "SplRW" : 40}

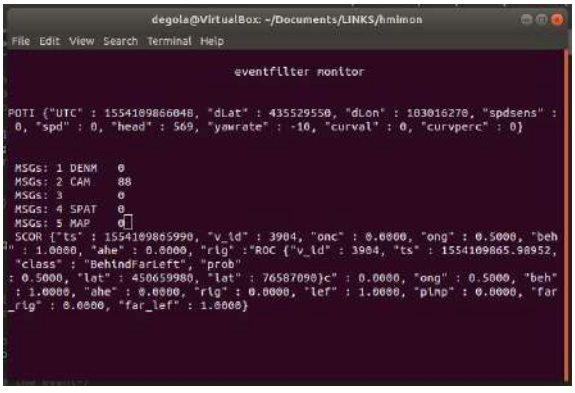
    - i messaggi relativi alle biciclette connesse segnalano la caduta del ciclista e la sua distanza dall'egovehicle;

    HMI {"FallenBicyclistPresent" : 1, "FallenBicyclistLongDistance" : 43, "FallenBicyclistDistance" : 43}

    - i messaggi relativi a eventi semaforici segnalano la presenza di un semaforo connesso, la sua fase corrente, la sua distanza dall'egovehicle, il tempo mancante alla fase successiva e la presenza di un pedone sulla strada.

    HMI {"TRL present" : 1, "curPh" : 2, "stopdist" : 68, "nxtchangeIn" : 94, "pedestr" : 1}

Poiché i messaggi sono visualizzati a video così come il programma li riceve dalla OBU è complicato eseguire operazioni di debug sul comportamento dei veicoli e dei sensori, non avendo un'immagine chiara di ciò che sta accadendo in un certo momento e dovendo necessariamente lavorare su file di log.



Il programma hmimon in esecuzione

3. Analisi delle esigenze

Il gruppo di ricerca necessita di una visualizzazione chiara di ciò che i sensori e i veicoli stanno comunicando, per individuare i bachi più facilmente e procedere più velocemente con lo sviluppo della On-Board Unit. Questa interfaccia uomo-macchina o HMI (Human-Machine Interface) sarà poi utilizzata a bordo dei veicoli connessi e a guida autonoma, essendo sempre presente per motivi di sicurezza un guidatore umano in grado di agire sui comandi dell'autovettura, e deve basarsi sui messaggi introdotti nel capitolo precedente.

Inizialmente l'HMI verrà utilizzata su PC, ma successivamente dovrà essere visualizzabile anche su dispositivi mobili. Deve quindi poter essere utilizzata su PC, smartphone e tablet.

Nella prima fase del tirocinio si è cercato di capire come avrebbe dovuto essere sviluppata l'HMI per soddisfare al meglio le richieste dell'azienda. Sono possibili diverse soluzioni software, principalmente lo sviluppo come applicazione nativa, Progressive Web App o applicazione ibrida:

- Le applicazioni native o native app rappresentano la maggior parte delle applicazioni scaricate e utilizzate quotidianamente. Sono sviluppate per uno specifico sistema operativo, con vari linguaggi di programmazione e vengono installate tramite lo store del sistema operativo utilizzato dall'utente interessato. Essendo sviluppate specificatamente per un sistema operativo sono veloci e affidabili, hanno accesso a tutte le funzionalità del dispositivo e dopo l'installazione possono funzionare anche senza una connessione ad Internet.
- Le Progressive Web App sono applicazioni sviluppate come normali pagine web, ma che possono essere installate sul dispositivo dell'utente e utilizzate in modo simile alle applicazioni native. Il codice sorgente è condiviso da tutte le piattaforme, ma è necessaria una connessione ad Internet almeno al primo avvio. Si tratta di una tecnologia relativamente recente, ma che è stata adottata con successo da importanti aziende per la propria applicazione mobile.
- Le applicazioni ibride sono una soluzione intermedia tra native app e web app, in quanto il codice sorgente è lo stesso per ogni sistema operativo (solitamente sono sviluppate come applicazioni web), ma devono essere scaricate dallo store del proprio dispositivo e hanno quindi accesso alle sue funzionalità. Sono spesso realizzate tramite appositi framework, come Cordova o Ionic.

La visualizzazione verrà utilizzata su dispositivi e sistemi operativi diversi e non essendoci in 250 ore il tempo materiale per sviluppare più di una applicazione la soluzione delle native app è stata subito scartata. Inizialmente il gruppo di ricerca era orientato verso le applicazioni ibride, ma queste tendono a essere meno performanti delle applicazioni native e devono comunque essere installate sul dispositivo dell'utente. Framework come Apache Cordova incapsulano il codice

HTML, CSS3 e JavaScript in applicazioni disponibili per molti sistemi operativi, anche poco diffusi, e permettono di utilizzare l'hardware del dispositivo. Tuttavia, il risultato finale sarebbe simile a quello ottenuto con una Progressive Web App e richiedono in aggiunta il download dell'applicazione dallo store e l'utilizzo del framework, che complicherebbero il progetto e renderebbero l'esperienza d'uso meno immediata. Si decide quindi di approfondire le Progressive Web App e di provare a sviluppare l'interfaccia con questa tecnologia.

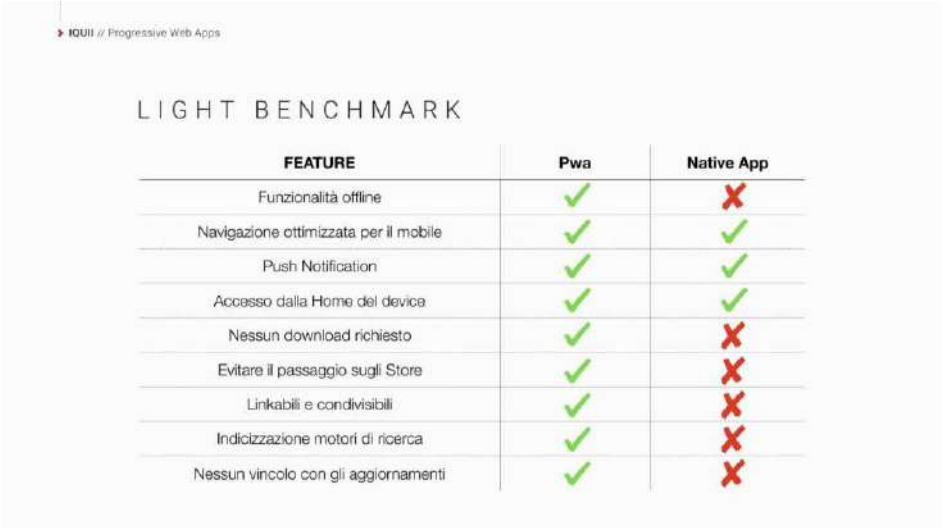
3.1. Le Progressive Web App

Il termine Progressive Web App è stato utilizzato per la prima volta nel 2015 dagli sviluppatori di Google per descrivere applicazioni web che, grazie alle funzionalità offerte dai browser moderni, come i Service Worker e gli App Manifest, offrono un'esperienza d'uso simile a un'applicazione nativa. Si definiscono "progressive" perché aggiungono funzioni alle normali pagine web in base a quelle offerte del dispositivo e progressivamente si comportano sempre più come applicazioni mobili.

Secondo le linee guida di Google, le applicazioni web si possono definire Progressive Web App se sono:

- progressive, cioè funzionano per ogni utente, a prescindere dal browser scelto perché sono costruite alla base con principi di miglioramento progressivo;
- app-like**,** cioè si comportano con l'utente come se fossero delle app native, in termini di interazione e navigazione;
- responsive, cioè si adattano alle varie dimensioni dello schermo: desktop, mobile, tablet, o dimensioni che potranno in seguito rendersi disponibili;
- sicure, dato che vengono esposte su protocollo HTTPS per evitare che la connessione mostri informazioni o che i contenuti vengano alterati;
- aggiornate, poiché le informazioni sono sempre aggiornate grazie al processo di aggiornamento dei dati offerto dai Service Worker;
- ricercabili, cioè vengono identificate come "applicazioni" e vengono indicizzate dai motori di ricerca;
- riattivabili, cioè rendono facile la riattivazione dell'applicazione grazie a capacità quali le notifiche web;
- installabili, cioè consentono all'utente di "salvare" le app che considera più utili con la corrispondente icona sullo schermo del proprio terminale mobile (home screen) senza che si debba affrontare tutti i passaggi e problemi legati all'uso dell'app store;
- linkabili, cioè facilmente condivisibili tramite l'URL senza complesse installazioni;
- offline, in quanto i Service Workers consentono di far funzionare l'applicazione anche in mancanza di connessione o con connessioni di bassa qualità.

Le Progressive Web App sono attualmente supportate completamente dai browser per dispositivi mobile più diffusi, mentre sono supportate parzialmente su desktop (tutte le funzionalità sono disponibili solamente su Google Chrome). I browser identificano un sito web come una Progressive Web App se questo è esposto da una sorgente sicura tramite TLS, può essere caricato anche quando il dispositivo dell'utente è offline (cioè possiede un Service Worker), ha un Web App Manifest con le proprietà chiave e un'icona grande almeno 144x144 pixel in formato PNG.



Confronto tra Progressive Web App e applicazioni native

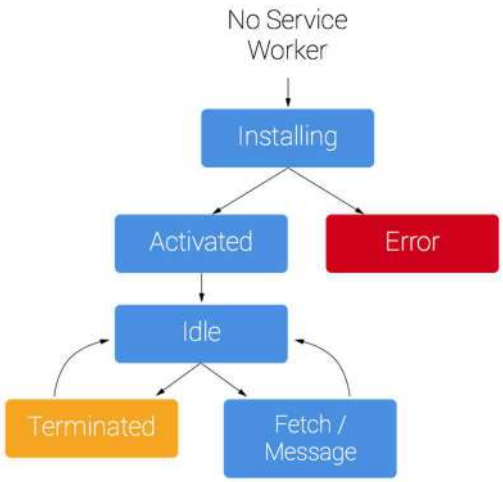
Come accennato in precedenza, le Progressive Web App si basano principalmente su due tecnologie implementate nei browser moderni: i Web App Manifest e i Service Worker.

Un **Web App Manifest** è un file in formato JSON contenente i metadata associati con l'applicazione web, per descriverla al browser e determinarne il comportamento all'installazione sul dispositivo dell'utente. Un manifesto è necessario per permettere all'utente di installare la Progressive Web App.

Per l'interfaccia sviluppata durante il tirocinio, intitolata HMI Visualizer, nel Web App Manifest si è indicato il nome, una breve descrizione, delle icone di diversa dimensione, la lingua principale dell'applicazione, lo `start_url` (l'URL della pagina visualizzata all'apertura dell'applicazione), il `background_color` (il colore della schermata di avvio), la modalità di visualizzazione (con `"display" : standalone` gli elementi appartenenti al browser sono nascosti, come in un'applicazione nativa), l'orientamento dello schermo, il gruppo di URL appartenenti all'applicazione e il `theme_color` (il colore degli elementi del browser e del sistema operativo relativi al sito web).

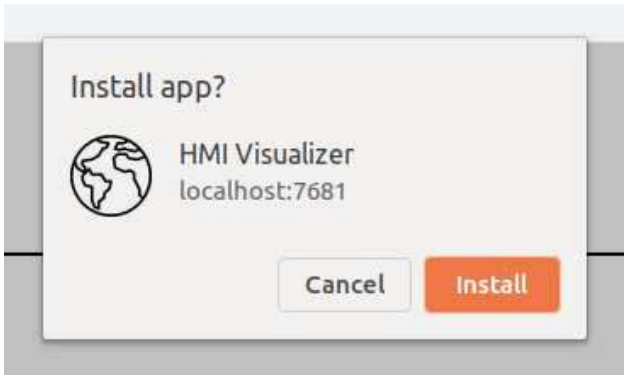
Un **Service Worker** è uno script JavaScript che funziona come un proxy di rete per gestire le richieste HTTP, operando in parallelo alla pagina web principale. In questo modo può gestire una o più cache e far funzionare la pagina web anche senza una connessione Internet. I Service Worker implementano inoltre funzioni come le notifiche push e la sincronizzazione in background e ne implementeranno altre in futuro.

Il ciclo di vita di un Service Worker è separato da quello della pagina web e viene registrato nel codice JavaScript principale. In questo modo il browser inizia l'installazione del Service Worker in background, salvando in cache i file richiesti. Se tutti i file sono salvati con successo il Service Worker risulta installato, altrimenti il processo viene ripetuto al caricamento successivo. Segue poi una fase di attivazione, in cui si può aggiornare il Service Worker, e a questo punto funziona da proxy o viene terminato per liberare memoria.



Il ciclo di vita di un Service Worker

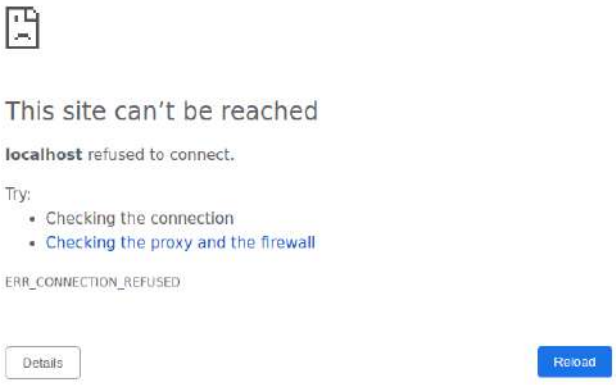
Se i Service Worker non sono supportati dal browser utilizzato o se la connessione non è sicura, il Service Worker non viene installato e l'applicazione è gestita come una pagina web tradizionale. Se installato correttamente, insieme al Web App Manifest permette l'installazione dell'applicazione sul dispositivo, che viene gestita diversamente da ogni browser. Google Chrome, il primo browser ad aver supportato le Progressive Web App, su mobile crea una WebAPK, gestita a tutti gli effetti come un'applicazione nativa, mentre su desktop l'applicazione è aperta in un'apposita finestra del browser. Se tutti i requisiti sono soddisfatti, il browser avvia uno specifico evento, detto `beforeinstallprompt`, il quale è utilizzato per avvisare l'utente che l'applicazione può essere installata. Nel nostro caso viene mostrato un bottone nella pagina web e al click si chiede all'utente se desidera installare l'applicazione. Se l'utente accetta, questa viene installata e diventa disponibile tra gli applicativi presenti sul dispositivo.



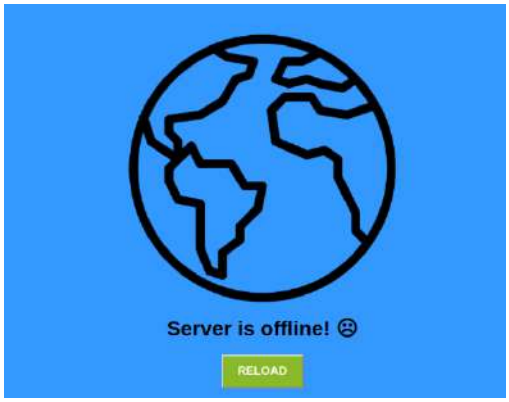
Il banner di installazione di Google Chrome

L'applicazione installata su Ubuntu 18.04

Per quanto riguarda il Service Worker, alla sua installazione si creano tre cache in sequenza, una per le immagini utilizzate nell'interfaccia, una per le pagine HTML e i fogli di stile per le pagine di errore e una per le librerie JavaScript. All'attivazione il Service Worker non subisce nessuna modifica, mentre quando riceve una richiesta dalla pagina web restituisce immediatamente l'elemento richiesto se si trova in cache, altrimenti tenta di recuperarlo dal server. Se rileva errori dovuti alla mancanza di connessione restituisce la pagina di errore personalizzabile, che viene quindi visualizzata. Se il browser è connesso ma l'elemento non è presente sul server, restituisce di default la pagina relativa all'errore HTTP 404 Not Found.



Prima dell'installazione del Service Worker



Dopo l'installazione del Service Worker

Il codice relativo alla realizzazione della Progressive Web App è riportato in appendice.

4. Il server WebSocket

L'interfaccia deve elaborare in tempo reale i dati che la OBU riceve dai sensori IoT e dagli altri veicoli. È quindi necessario inviare i messaggi ricevuti dall'hmimon introdotto nel paragrafo 2.4. a un apposito server, in grado a sua volta di trasmetterli alla pagina web, aperta anche in più finestre contemporaneamente. Per realizzare il server si utilizza il protocollo WebSocket e la libreria C libwebsockets, dopo avere opportunatamente modificato il programma hmimon. Oltre a servire l'applicazione oggetto del tirocinio il server è stato condiviso con un altro studente tirocinante al lavoro su un diverso tipo di visualizzazione.

4.1. Il protocollo WebSocket

WebSocket è un protocollo di comunicazione web standardizzato dall'IETF (Internet Engineering Task Force) nel 2011 come RFC 6455 (Request for Comments). Si posiziona al livello 7 del modello OSI e si basa su una connessione TCP come il protocollo HTTP, ma diversamente da quest'ultimo permette a un client e a un server di scambiarsi informazioni in ogni momento senza una richiesta esplicita una volta stabilita la connessione. In questo modo il server può inviare dati al client non appena questi sono disponibili, comunicando su un canale full-duplex basato su una singola connessione. È comunque compatibile con l'HTTP, in quanto utilizza le stesse porte e un simile meccanismo di handshake, ed è supportato da tutti i browser più diffusi.

Sul lato client, le WebSocket API sono standardizzate dal W3C (World Wide Web Consortium), supportate da molti linguaggi di programmazione e permettono di comunicare su una connessione WebSocket.

4.2. Lo sviluppo del server e la libreria libwebsockets

Per realizzare il server WebSocket è stata utilizzata la libreria C libwebsockets, molto leggera e quindi adatta ad essere installata sul server interno della On-Board Unit, sul quale è disponibile un compilatore C. La maggior parte delle funzioni utilizzate sono proprietarie di libwebsockets e la documentazione non è particolarmente dettagliata, ma fortunatamente gli sviluppatori hanno reso disponibili circa 50 semplici esempi liberamente modificabili, da utilizzare come punto di partenza per lo sviluppo del server e per capire il funzionamento della libreria. libwebsockets permette inoltre di implementare una connessione sicura su TLS, ma questa non è stata sviluppata durante il tirocinio per mancanza di tempo e verrà aggiunta in seguito.

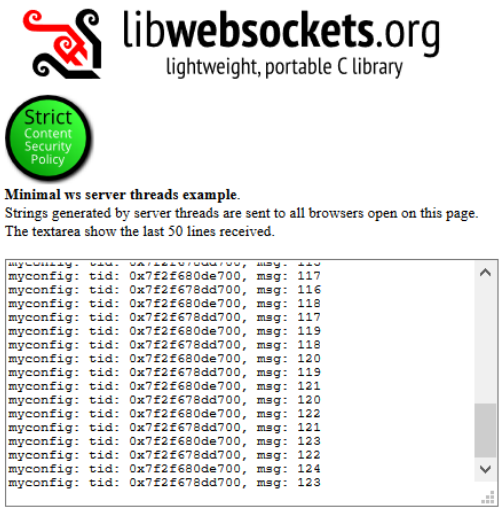
Generalmente, il comportamento di un server WebSocket implementato con libwebsocket è descritto in due file .c, chiamati server.c e protocol.c. Nel file server.c si descrive l'inizializzazione dell server vero e proprio indicandone i parametri come la porta di funzionamento, i direttori da cui recuperare i file da servire, il protocollo di funzionamento e una serie di opzioni. Nel file protocol.c viene invece descritto il comportamento del server tramite una funzione di callback, chiamata ad ogni evento e che esegue operazioni diverse a seconda del tipo di evento, indicato

da un `enum lws_callback_reason` la cui descrizione inizia per `LWS_CALLBACK`. I principali eventi e fasi di un server libwebsockets possono essere descritti come segue:

- `LWS_CALLBACK_PROTOCOL_INIT`: chiamato una sola volta all'esecuzione del server, per recuperare le informazioni definite in server.c e configurare il server;
- `LWS_CALLBACK_ESTABLISHED`: chiamato una volta per ogni nuovo client connesso, per memorizzarlo in una apposita coda;
- `LWS_CALLBACK_RECEIVE`: chiamato ogni volta che il server riceve nuovi dati da inviare ai client. Solitamente a questo punto si genera un evento `LWS_CALLBACK_SERVER_WRITEABLE`;
- `LWS_CALLBACK_SERVER_WRITEABLE`: chiamato dall'evento precedente, qui il server invia i dati disponibili a tutti i client connessi;
- `LWS_CALLBACK_CLOSED`: chiamato quando un client si disconnette, questo viene rimosso dalla coda;
- `LWS_CALLBACK_PROTOCOL_DESTROY`: chiamato una sola volta quando il server viene terminato per liberare la memoria.

Si assegna inoltre un nome al protocollo, necessario per la connessione dei client.

Nel nostro caso i messaggi da inviare al browser devono essere recuperati dall'hmimon, già in esecuzione parallelamente al server. Dopo aver analizzato gli esempi, il più vicino a questo comportamento risulta essere quello chiamato minimal-ws-server-threads, in cui nella fase di inizializzazione il server avvia due thread asincroni, i quali generano una stringa ogni 100 millisecondi e la aggiungono ad un buffer circolare segnalando al server che sono disponibili nuovi dati da inviare ai client. Differentemente dal comportamento descritto in precedenza il server



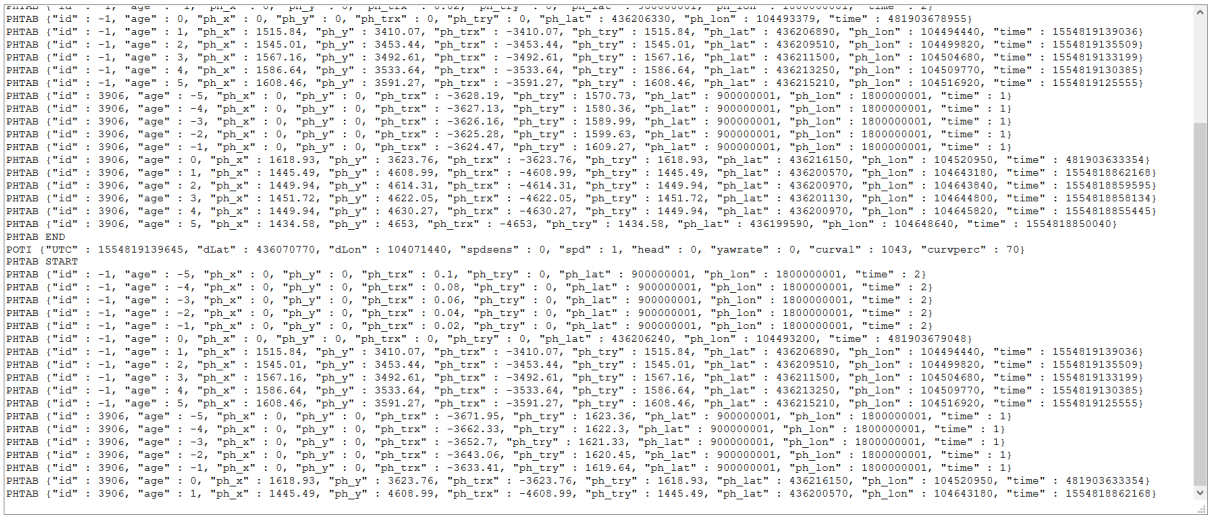
Il programma minimal-ws-server-threads in esecuzione

non riceve i dati dai client e quindi non si origina un evento `LWS_CALLBACK_RECEIVE`, ma ogni thread dopo aver scritto il messaggio causa un evento `LWS_CALLBACK_EVENT_WAIT_CANCELLED` con lo stesso comportamento dell'evento `LWS_CALLBACK_RECEIVE`.

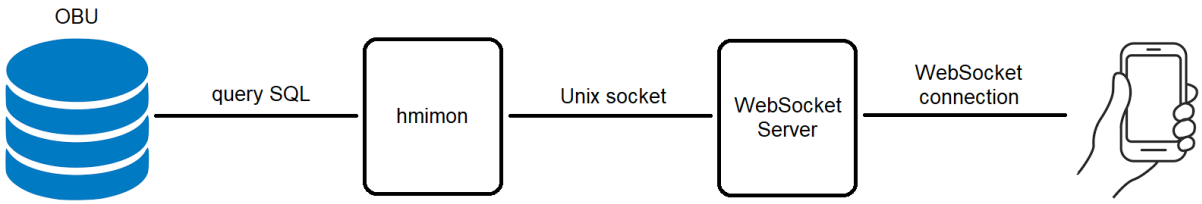
Prima di procedere con la modifica del server è necessario modificare l'hmimon in modo da creare un canale di comunicazione tra i due eseguibili su cui trasmettere e ricevere i messaggi. Per fare ciò sono stati utilizzati dei socket, un argomento che non viene affrontato nei corsi della Laurea triennale in Ingegneria Informatica, dapprima su connessione TCP e poi in dominio Unix. In questo modo, quando l'hmimon viene lanciato con uno specifico parametro su linea di comando, i messaggi ricevuti dal server della OBU non sono stampati a video ma sono scritti su un socket Unix con la funzione **`write`** e possono essere letti con la funzione **`read`** da un altro programma che conosce il suo indirizzo. Per interpretare correttamente i dati inviati la scrittura di ogni messaggio è preceduta dalla sua lunghezza in formato **`uint16_t`**.

A questo punto è stato quindi possibile modificare il protocollo di funzionamento del server. Dato che in fase di sviluppo è particolarmente utile lavorare su file di log il server può essere avviato con diversi parametri e legge dal socket o da un log a seconda di quello ricevuto.

Il file server.c non è stato modificato, tranne per i direttori contenenti i file della pagina web e per una fase di verifica del parametro ricevuto con modifica di alcuni flag. Per il file protocol.c, in fase di inizializzazione viene avviato un solo thread che legge i messaggi dal file di log o da socket, memorizzandoli in un buffer circolare e avvisando il server che sono presenti messaggi da inviare ai client. Nonostante si usi un solo thread si mantengono alcune istruzioni relative alla mutua esclusione tra thread presenti in origine, per assicurare la corretta trasmissione dei dati ai client.



Un esempio di messaggi inviati dal server WebSocket e visualizzati da un client



Schema di funzionamento

5. L'interfaccia grafica

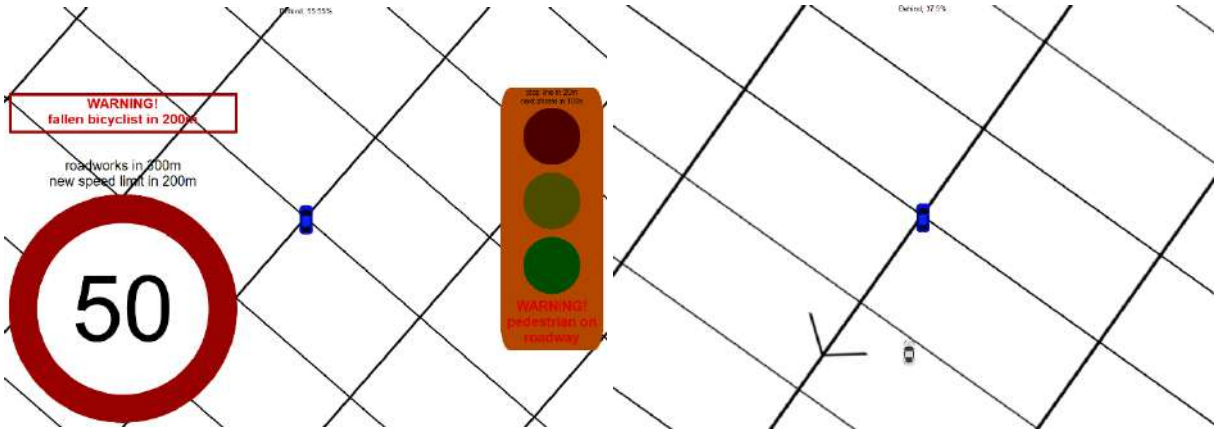
La maggior parte delle 250 ore di tirocinio è stata dedicata allo sviluppo dell'interfaccia grafica vera e propria, realizzata come una pagina web tradizionale. L'idea iniziale del gruppo di ricerca è sviluppare una visualizzazione in stile navigatore satellitare, cioè con l'egovehicle al centro dello schermo e con la mappa sullo sfondo che ruota in base alla sua direzione. In aggiunta si dovrebbe indicare la posizione degli altri veicoli e gli eventi stradali ricevuti dai dispositivi IoT. Per realizzare una grafica con un comportamento simile sono disponibili diverse librerie JavaScript, come D3.js e CanvasJS, ma forniscono funzionalità avanzate e non necessarie per quello che richiede l'azienda. L'interfaccia è quindi sviluppata in linguaggio JavaScript semplice per mantenerla leggera, utilizzando solamente la libreria JQuery per alcune animazioni.

Il client si connette al server WebSocket tramite una funzione presente negli esempi di libwebsockets, la quale sostituisce la sottostringa "http" dell'URL della pagina con "ws" ("https" con "wss" se si è su una connessione sicura) e apre una connessione WebSockets con il protocollo implementato nel server.

Per ogni messaggio che il server invia sulla connessione WebSocket il client memorizza la prima sottostringa, cioè la sua descrizione, e ne analizza il resto trattandolo come un JSON. La struttura della pagina è gestita da semplice codice HTML e CSS, il quale viene modificato dal codice JavaScript quando la connessione WebSocket riceve dei messaggi.

5.1. La prima versione

Per la prima versione si è usato come sfondo una griglia, con l'idea di implementare la mappa in un secondo momento. Nelle immagini qui riportate il veicolo azzurro rappresenta l'egovehicle, mentre quelli bianchi rappresentano gli altri veicoli.



La prima versione dell'HMI, in presenza di eventi stradali e di altri veicoli

Quando si riceve un messaggio POTI, cioè relativo all'egovehicle, si memorizza la posizione corrente del veicolo e si ruota la griglia in base alla sua direzione. Ciò avviene ogni tre messaggi ricevuti per risparmiare capacità di calcolo. Si aggiorna inoltre l'informazione sul tempo attuale, per gestire l'eliminazione di veicoli di cui non si ricevono informazioni da tempo.

Per la gestione della posizione degli altri veicoli ci si è basati sui messaggi PHTAB, in quanto, come accennato nel paragrafo 2.4., i messaggi ROC non erano interpretabili correttamente e non era possibile risolvere immediatamente il problema. I messaggi PHTAB, usati solitamente per debug, contengono informazioni sulla storia dei veicoli (path history) e previsioni sulle posizioni future. Tra le varie informazioni indicano la posizione assoluta del veicolo in coordinate geografiche e la distanza in metri dall'egovehicle, nei campi `ph_trx` e `ph_try`. Inizialmente si sono usati questi ultimi per inserire un'icona sullo schermo per ogni veicolo, considerando che l'egovehicle si trova al centro, ma non essendo molto stabili si è preferito utilizzare le coordinate assolute, considerando la differenza con le coordinate dell'egovehicle e la rotazione del sistema di riferimento. Per ogni nuovo veicolo si aggiunge un `div` HTML e se ne modificano le proprietà CSS. Dato che i PHTAB non danno informazione sulla direzione dei veicoli la loro icona è sempre diretta verso l'alto dello schermo.

```
var x = obj[‘ph_lon’] - curlon; var y = obj[‘ph_lat’] - curlat;     //latitude and longitude of egovehicle
var X = x*Math.cos(curh_rad) + y*Math.sin(curh_rad);
var Y = -x*Math.sin(curh_rad) + y*Math.cos(curh_rad);         //position according to egovehicle’s heading
```

Per ogni evento HMI è stato progettato un elemento HTML cha si aggiorna a ogni nuovo messaggio. Ad esempio, per gli eventi semaforici si accende la luce corrispondente alla fase attuale e i limiti di velocità vengono mostrati nello stile di un segnale stradale. In questo modo l'utente può rendersi conto in tempo reale della posizione degli altri veicoli e della presenza di eventi stradali.

Il passo successivo per questa visualizzazione è l'aggiunta della mappa sullo sfondo, possibile con diverse librerie JavaScript. La più diffusa libreria open source è Leaflet, molto leggera ma senza le funzionalità richieste dall'azienda come la rotazione della mappa e la visione in prospettiva. Altre due librerie molto utilizzate sono Mapbox.js e OpenLayers. La prima è molto completa ma dopo averla testata è risultata pesante e il caricamento della mappa non era abbastanza veloce, mentre OpenLayers è più leggera e ha la possibilità di ruotare la mappa, ma per la visuale prospettica si deve ricorrere a librerie aggiuntive.

5.2. La seconda versione

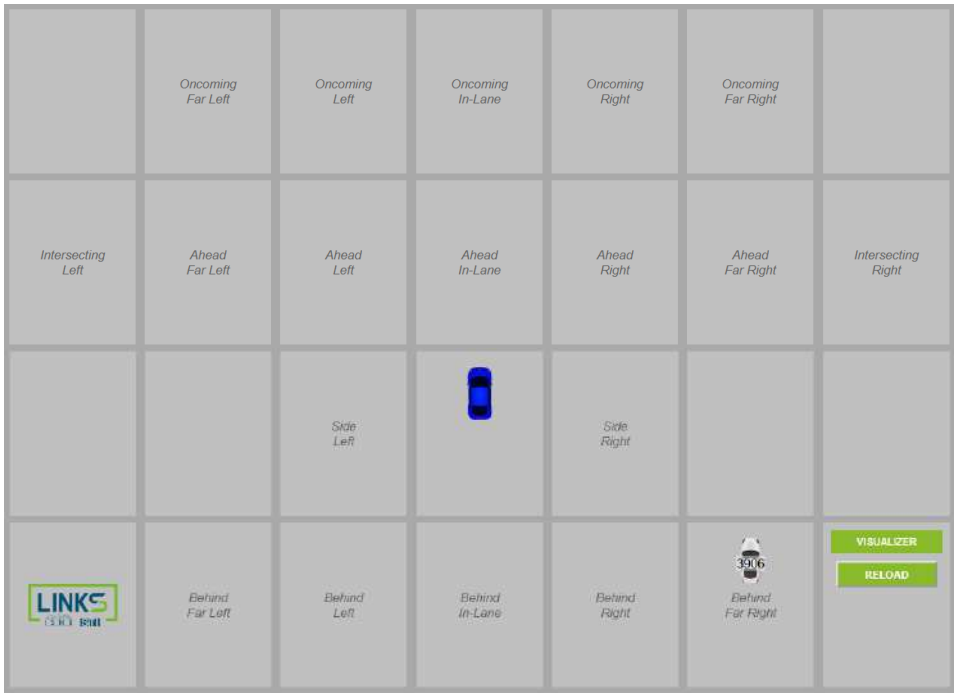
Dopo una serie di riunioni i tutori aziendali hanno deciso di indirizzare lo sviluppo verso un'interfaccia uomo-macchina più tradizionale, rimuovendo la mappa e mostrando sullo schermo le informazioni su veicoli ed eventi. Il lavoro svolto in precedenza è comunque stato utile in quanto mi ha permesso di prendere confidenza con tecnologie che non avevo mai utilizzato prima.

In questa visualizzazione tutte le informazioni sono disponibili sullo schermo per debug. L'egovehicle è rappresentato su una strada stilizzata che segue la reale curvatura, mentre per semplicità non sono rappresentate le icone per gli altri veicoli. Gli eventi stradali sono rappresentati in modo simile alla versione precedente, in un layout regolare realizzato con tabelle HTML.

Quando il client riceve messaggi POTI aggiorna le informazioni stampate a video e la curvatura della strada, secondo il campo `curval`. Dato che la posizione degli altri veicoli non è importante le informazioni sono recuperate dai messaggi ROC. Quando si ricevono messaggi HMI viene mostrato il `div` HTML corrispondente e le informazioni sono aggiornate. Tutto avviene ogni tre messaggi ricevuti di uno stesso tipo per risparmiare capacità di calcolo. Le informazioni risultano comunque corrette e interpretabili dall'utente, che in questo modo ha meno distrazioni rispetto alla versione precedente e può cogliere immediatamente gli eventi importanti.

La seconda versione dell'HMI

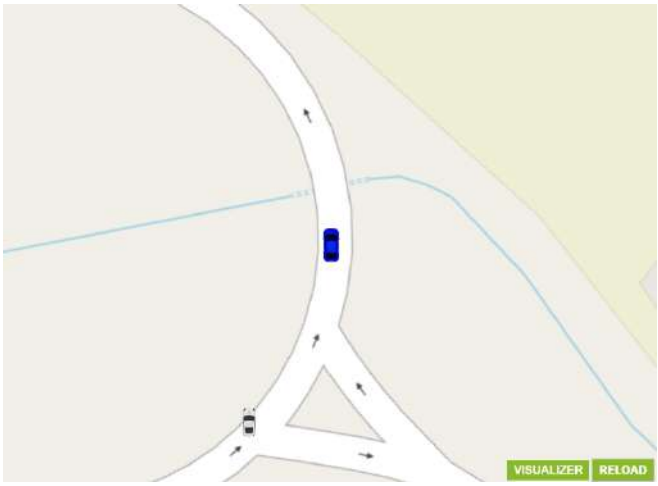
Il gruppo di ricerca ha poi richiesto lo sviluppo di un secondo tipo di visualizzazione, necessario per verificare il loro algoritmo per la stima della posizione relativa dei veicoli rispetto all'egovehicle. L'informazione è contenuta nel campo `class` dei messaggi ROC, i quali contengono anche la probabilità che l'informazione sia corretta. La visualizzazione si basa sulle `grid` CSS, dividendo lo schermo in porzioni uguali, con l'egovehicle rappresentato al centro e gli altri veicoli nella posizione corrente. I possibili valori della proprietà `class`, oltre a quello denominato Unclassified se non è possibile stimare la posizione, sono rappresentati nell'immagine successiva.



La visualizzazione a griglia

Nel codice JavaScript, ogni volta che il client riceve un messaggio ROC si crea un nuovo elemento se il veicolo non è ancora presente, altrimenti si aggiorna quello vecchio spostandolo in un'altra casella se ha cambiato posizione. Se la posizione risulta Unclassified, si stampa un messaggio nella casella centrale.

Proseguendo il lavoro sulla prima versione dell'interfaccia, si è inoltre sviluppata una visualizzazione a mappa basata sul framework OpenLayers. L'egovehicle è sempre al centro della mappa, e ad ogni messaggio POTI se ne aggiorna il centro e la rotazione, basandosi sulle sue coordinate e direzione. Il gruppo di ricerca ha poi risolto la doppia chiave dei messaggi ROC e le coordinate degli altri veicoli possono essere interpretate correttamente e utilizzate per rappresentarli sulla mappa. Non si hanno ancora informazioni sulla loro direzione.



La visualizzazione a mappa

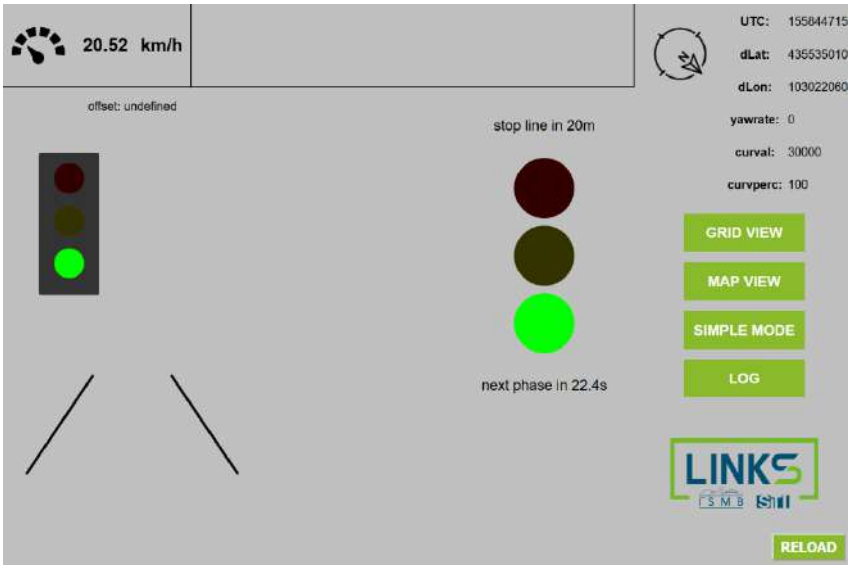
Ogni visualizzazione utilizza diverso codice HTML, CSS e Javascript, ed è possibile passare da una visualizzazione all'altra con degli appositi bottoni, come visibile nelle immagini precedenti. Sono poi presenti un bottone per aggiornare la pagina se si apre una nuova connessione WebSocket e uno per installare la Progressive Web App se supportata dal browser.

I bottoni della visualizzazione a mappa sul browser Google Chrome, in ordine per tornare alla visualizzazione principale, ricaricare la pagina web o installare l'applicazione

5.3. La versione finale

L'interfaccia uomo-macchina descritta nel paragrafo precedente è un'interfaccia completa che può essere utilizzata per verificare il comportamento di veicoli connessi e dispositivi IoT. Nelle ultime settimane di tirocinio il gruppo di ricerca ha richiesto di sostituire la strada stilizzata con una visuale degli eventi in prospettiva, così da segnalare all'utente la loro distanza in modo più immediato. Questa è stata realizzata introducendo nuovi elementi HTML che si aggiornano insieme a quelli già presenti e la distanza dall'evento è resa modificando la dimensione del segnale dell'evento tramite la proprietà CSS `scale`, in relazione alla distanza effettiva.

È stato inoltre aggiunto un campo `offset` nei messaggi POTI, usato per debug.



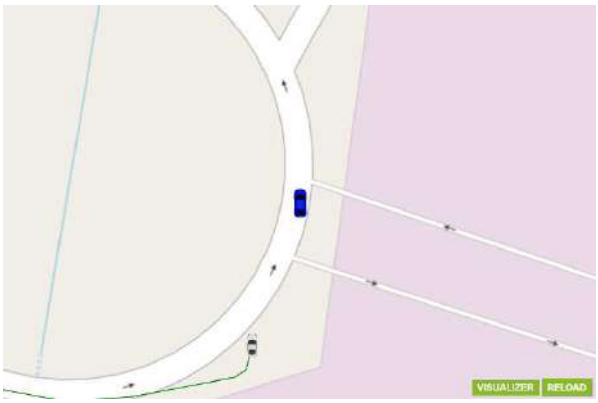
La versione finale dell'HMI, con eventi semaforici

La visualizzazione a griglia non è stata modificata rispetto alla versione precedente, mentre per la visualizzazione a mappa è stato introdotto un nuovo tipo di messaggio, intitolato ALTER, simile ai messaggi ROC ma contenente anche la rotazione rispetto a nord del veicolo in decimi di grado. In questo modo è possibile ruotare anche le icone degli altri veicoli sulla mappa, rispetto alla direzione dell'egovehicle.

ALTER {“v_id” : 3904, “UTC” : 442565416913344916, “stty” : 5, “lat” : 435548360, “lon” : 103037460, “head” : 1319, “spd” : 453}

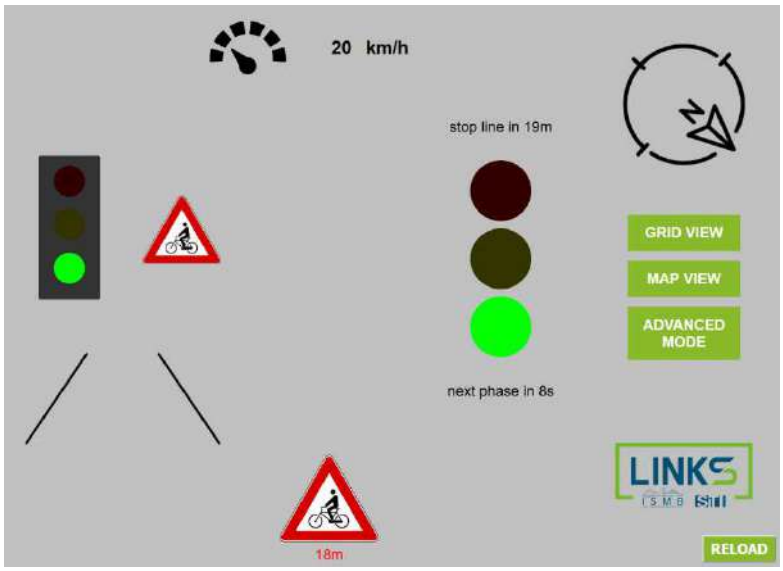
Per scopi di debug sono stati poi introdotti dei segmenti OpenLayers che collegano le posizioni previste nei messaggi PHTAB. Questa funzione è stata introdotta gli ultimi giorni e lo sviluppo è stato completato dal gruppo di ricerca.

La rotazione dei veicoli con i messaggi ALTER



I segmenti per i messaggi PHTAB

È stata poi introdotta una visualizzazione semplificata, simile a quella principale ma senza informazioni di debug, pensata per essere utilizzata da un utente finale. Le due visualizzazioni hanno molti elementi in comune perciò il codice è condiviso, con un flag che segnala se si sta utilizzando la visualizzazione completa o quella semplificata.

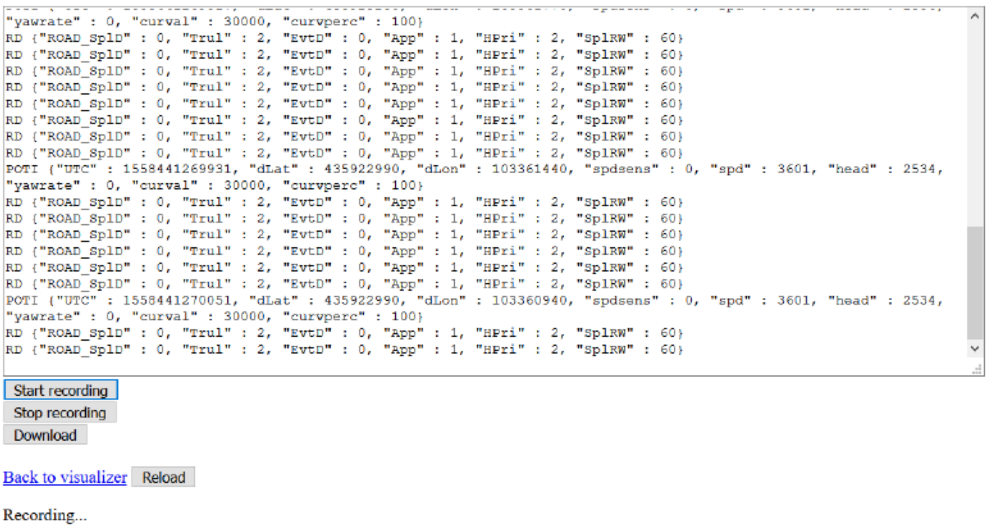


La visualizzazione semplificata, con eventi semaforici e biciclette connesse

Le due visualizzazioni sono state inoltre adattate a schermi in verticale, come quelli di smartphone e tablet, per permettere il corretto utilizzo dell’applicazione su ogni tipo di dispositivo.

Le due visualizzazioni in verticale, con eventi stradali, eventi semaforici e attraversamenti pedonali

Infine, per necessità di debug è stata introdotta la possibilità di visualizzare su browser i messaggi ricevuti dalla OBU e quindi dal server WebSocket. Qui l'utente può registrare i messaggi a cui è interessato e scaricarli come un file di testo.



Registrazione dei messaggi, cliccando su Download il browser avvia lo scaricamento del file di log

6. Conclusione

Al termine del tirocinio il server WebSocket e i file necessari al funzionamento dell'applicazione sono stati installati con successo sulla On-Board Unit e testati in laboratorio con dispositivi IoT reali e simulati. L'Area di Ricerca Multi-Layer Wireless Solutions della Fondazione LINKS ha quindi a disposizione un'interfaccia uomo-macchina completa e funzionante con diverse modalità di visualizzazione, per supportare lo sviluppo di veicoli a guida autonoma in un ambiente connesso con sensori IoT. L'interfaccia può sicuramente essere migliorata sia graficamente che nelle prestazioni, ma rappresenta un ottimo punto di partenza ed è utilizzabile fin da subito. L'azienda si ritiene quindi soddisfatta di ciò che è stato realizzato in 250 ore, corrispondenti a poco più di tre mesi di lavoro.

In conclusione, l'esperienza di tirocinio presso la Fondazione LINKS è stata decisamente positiva, in quanto ho avuto modo di conoscere tecnologie che non vengono affrontate nei corsi della laurea triennale in Ingegneria Informatica presso il Politecnico di Torino e che non avrei utilizzato altrimenti, come il protocollo WebSocket o le applicazioni web, e di svilupparne altre che conoscevo solo in parte, come il linguaggio JavaScript.

Infine, ho avuto modo di prendere contatto con il mondo del lavoro grazie a un importante centro di ricerca con un'ottima reputazione, in un ambiente informale e dinamico. Gli ingegneri Daniele Brevi, Guido Alejandro Gavilanes Castillo e Edoardo Bonetto sono sempre stati disponibili per supporto tecnico e di progetto e mi hanno permesso di comprendere il processo completo di sviluppo di un'applicazione, una competenza che difficilmente si può acquisire in un ambiente diverso da quello lavorativo.

Appendice

Il Web app manifest

```
{
    "short_name": "HMIView",
    "name": "HMI Visualizer",
    "description": "Visualizer for LINKS Fundation's OBUs",
    "icons": [
        {
                "src": "../images/icons-192.png",
                "type": "image/png",
                "sizes": "192x192"
        },
        {
             "src": "../images/icons-512.png",
             "type": "image/png",
             "sizes": "512x512"
        }
    ],
    "lang": "en-US",
    "start_url": "/",
    "background_color": "#000080",
    "display": "standalone",
    "orientation": "any",
    "scope": "/",
    "theme_color": "#000080"
}
```

La registrazione del Service Worker

```
if ('serviceWorker' in navigator) {
    navigator.serviceWorker.register('sw.js', {scope: '/'}).then(function(registration) {
             console.log('service worker successfully installed. scope:', registration.scope);
    }).catch(function(error) {
             console.log('service worker installation failed:', error);
    });
}

/*add to homescreen*/
let deferredPrompt;
const addBtn = document.querySelector('.add-button');
addBtn.style.display = 'none';

window.addEventListener('beforeinstallprompt', (e) => {
  // Prevent Chrome 67 and earlier from automatically showing the prompt
  e.preventDefault();
  // Stash the event so it can be triggered later.
  deferredPrompt = e;
  // Update UI to notify the user they can add to home screen
  addBtn.style.display = 'inline-block';

  addBtn.addEventListener('click', (e) => {
    // hide our user interface that shows our A2HS button
    addBtn.style.display = 'none';
    // Show the prompt
```

```
    deferredPrompt.prompt();
    // Wait for the user to respond to the prompt
    deferredPrompt.userChoice.then((choiceResult) => {
        if (choiceResult.outcome === 'accepted') {
          console.log('User accepted the A2HS prompt');
        } else {
          console.log('User dismissed the A2HS prompt');
        }
        deferredPrompt = null;
      });
  });
});
```

Il Service Worker

```
// SW installation, initialize caches
self.addEventListener('install', event => {
    event.waitUntil(
             caches.open('icons').then(cache => {
                     //console.log('caching...');
                     return cache.addAll([
                             //images' path
                     ])
                     .then(caches.open('pages').then(cache => {
                             return cache.addAll([
                                     //HTML and CSS files' path
                             ])
                             .then(caches.open('libs').then(cache => {
                                     return cache.addAll([
                                             //JS libraries' path
                                     ])
                                     .then(() => self.skipWaiting());
                             }))
                     }))
             }))
    console.log("Service Worker installed");
});

// Activate event
self.addEventListener('activate', event => {
    event.waitUntil(self.clients.claim());
    console.log("Service Worker active");
});

// Fetch event, if requested element is not cached get it from server, if there is an error return the
customised error page
self.addEventListener('fetch', event => {
    //console.log("Requested URL: " + event.request.url);
    event.respondWith(
        caches.match(event.request).then(response => {
            return response || fetch(event.request).catch(error => {
            //console.log(error);
            return caches.match(offlinePage);
            });
        })
    );
});
```

Bibliografia e sitografia

[1] https://linksfoundation.com/

[2] https://compagniadisanpaolo.it/ita/News/Nasce-la-Fondazione-Links

[3] http://www.ismb.it/Multi-Layer_Wireless_Solutions

[4] https://autopilot-project.eu/

[5] Brevi D. et al., ""Smart Roads" for AD cars: the AUTOPILOT Project in Livorno", https://autopilot-project.eu/wp-content/uploads/sites/16/2019/05/ITS19-Brainport-Paper-LivornoPS_Published.pdf

[6] https://www.mobiloud.com/blog/native-web-or-hybrid-apps/

[7] https://cordova.apache.org/

[8] https://developers.google.com/web/progressive-web-apps/

[9] https://medium.com/iquii/progressive-web-app-pwa-what-they-are-pros-and-cons-and-the-main-examples-on-the-market-318f4538c670

[10] Russel A., "What, Exactly, Makes Something A Progressive Web App?" https://infrequently.org/2016/09/what-exactly-makes-something-a-progressive-web-app/

[11] https://www.evemilano.com/progressive-web-app/

[12] https://developers.google.com/web/fundamentals/web-app-manifest/

[13] https://developers.google.com/web/fundamentals/primers/service-workers/

[14] https://developers.google.com/web/fundamentals/app-install-banners/

[15] Fette I. et al., "The WebSocket Protocol", https://tools.ietf.org/html/rfc6455

[16] https://libwebsockets.org/

[17] https://libwebsockets.org/lws-api-doc-master/html/group__usercb.html

[18] https://libwebsockets.org/git/libwebsockets/tree/minimal-examples/ws-server/minimal-ws-server-threads?h=v3.1-stable

[19] https://developer.mozilla.org/en-US/docs/Web/JavaScript

[20] https://jquery.com/

[21] https://leafletjs.com/

[22] https://docs.mapbox.com/mapbox.js/api/v3.2.0/

[23] https://openlayers.org/