POLITECNICO DI TORINO

Laurea Magistrale in Ingegneria Informatica

Laurea Magistrale

Progettazione ed implementazione di un sistema embedded per il rilevamento di danni ad un veicolo

Relatore
Prof. Giovanni Malnati

Candidato
Cristiano Palazzi
matricola 231775

Marzo 2019

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

Negli ultimi anni il numero di sinistri stradali è aumentato notevolmente, specialmente durante le manovre di parcheggio. La principale differenza tra gli incidenti che si verificano tra veicoli in movimento e gli incidenti che avvengono in fase di parcheggio è la dinamica dell'accaduto, la quale risulta più semplice da ricostruire sulla base della testimonianza dei conducenti. Ricostruire la dinamica di un incidente tra un veicolo che sta effettuando una manovra di parcheggio e le autovetture adiacenti è più complicato. Infatti, il proprietario che torna alla propria vettura dopo una sosta e nota un danno, non ha alcun modo di ricostruire l'accaduto, se non in presenza di eventuali testimoni. Queste sono le premesse dalle quali è partito il mio lavoro di tesi, che si pone come obiettivo quello di poter fornire al proprietario uno strumento per poter ricostruire la dinamica di una collisione avvenuta in sua assenza.

1.1 Analisi del problema

Una ricerca condotta in Germania dal Centro Tecnologico Allianz in collaborazione con Continental AG nel 2015 [1] attesta che il 40 % degli incidenti stradali avviene durante le fasi di parcheggio o sosta; dato calcolato sulla base di 3500 incidenti. Gli esperti ritengono che alla base di questo notevole dato ci sia il cambio di design delle vetture, il quale comporterebbe non solo un aumento delle dimensione del veicolo ma una diminuzione della visuale di guida disponibile. A differenza dei sinistri stradali che avvengono in movimento, in questi casi risulta molto più difficile l'individuazione del colpevole e la corretta ricostruzione della dinamica. I proprietari dei veicoli parcheggiati che subiscono danni possono solo avvalersi della testimonianza di passanti che forniscono una descrizione dell'accaduto ed eventualmente la targa del conducente colpevole. Va considerata, infatti, anche la spiacevole situazioni in cui il proprietario del veicolo incriminato si dilegui, disinteressandosi del danno arrecato, senza lasciare le proprie generalità per essere rintracciato. In questi casi la parte lesa non ha la possibilità di fornire alcune prova dell'accaduto alla compagnia assicurativa per ottenere un risarcimento. Infatti le più comuni polizze assicurative, come l'RC auto, non coprono queste eventualità ed è quindi necessario identificare il colpevole per ottenere un rimborso dalla sua compagnia assicurativa. E' anche possibile arricchire la propria assicurazione con ulteriori garanzie, come la polizza kasko che fornisce un risarcimento del danno a prescindere dalla responsabilità, ma

spesso hanno un costo molto elevato ed inoltre espongono le compagnie assicurative alla possibilità di truffa da parte dei clienti che la sottoscrivono.

1.2 Misure preventive

Queste motivazioni hanno portato le compagnia assicurative all'adozione di misure preventive, con lo scopo di fornire maggiori dettagli in caso di sinistro e di limitare le frodi: la scatola nera. La *insurance box* si basa sullo stesso principio delle scatole nere degli aerei e fornisce numerosi dati rilevati durante la guida, in modo da poter agevolare la ricostruzione della dinamica in caso di urto, evitando il rischio di imbattersi in una truffa da parte del conducente. La sua installazione è generalmente a carico della compagnia assicurativa e garantisce una riduzione sulla polizza RCA, in cambio di una serie di vantaggi di cui il cliente può usufruire. La scatola nera viene collegata alla batteria dei veicolo e le informazioni raccolte riguardano:

- Localizzazione
- Tempi di percorrenza
- Valori di accelerazione
- Informazioni relativi ad un sinistro e i sistemi di sicurezza che si sono attivati

Sebbene queste informazioni siano di grande utilità per l'assicurato, le scatole nere hanno una conoscenza totale delle sue abitudini ed hanno quindi un forte impatto sulla sua privacy, infatti tutti i dati vengono inviati alla compagnia assicurativa che può quindi monitorare il suo comportamento alla guida. Inoltre, vale la pena notare che le informazioni rilevate durante un sinistro potrebbero non essere sufficienti per la ricostruzione affidabile dell'accaduto e potrebbero portare ad un mancato risarcimento, da parte dell'assicurazione, anche in caso di ragione.

In conclusione, le scatole nere possono sicuramente fornire un valido supporto alle compagnie assicurative per esaminare la dinamica di un sinistro e all'assicurato che può essere tutelato al meglio e può sfruttare la tecnologia messa a disposizione, ad esempio in caso di furto il GPS della scatola semplificherebbe di molto la ricerca del veicolo. Il prezzo da pagare è rappresentato dalla quantità di dati raccolti che non è sotto il diretto controllo dell'utilizzatore, privandolo completamente della sua privacy.

1.3 Motivazione della tesi

In base a quanto emerso, la motivazione del mio lavoro di tesi è stata quella di fornire un valido strumento al proprietario di un veicolo parcheggiato per poter identificare la natura di eventuali collisioni avvenute in sua assenza. Queste ragioni mi hanno spinto ad analizzare un sistema che possa garantire gli stessi ed ulteriori vantaggi delle scatole nere, senza gli svantaggi che esse comportano.

1.4 Soluzione proposta

In questa sezione viene analizzato l'utilizzo del sistema e di come i dati che raccoglie possano essere usati dal proprietario del veicolo urtato per poter ricostruire la dinamica dell'incidente e identificare la vettura urtante. Le informazioni ottenute dal sistema potranno essere utilizzate come prova legale da fornire all'assicurazione per ottenere un risarcimento.

L'idea proposta si riferisce ad un sistema, pensato per essere utilizzato quando il veicolo è parcheggiato, in grado di registrare video e dati di accelerazioni in caso di incidente. Queste informazioni vengono continuamente registrate dal sistema, ma salvate nella memoria interna solo nell'eventualità di una collisione. L'utilizzo di un accelerometro per rilevare gli urti è stato preferito all'impiego di algoritmi di motion detection, che potrebbero fornire risultati errati sopratutto in ambienti affollati e rappresenta la condizione principale per il salvataggio dei dati nel sistema. In questo modo vengono memorizzati solo i video e i dati di accelerazione relativi all'urto che possono essere condivisi con la compagnia assicurativa come prova del danno subito dal proprio veicolo.

Lo scopo del sistema è registrare video e dati aggiuntivi come l'accelerazione, l'orario ed opzionalmente la localizzazione tramite GPS, quando il veicolo è parcheggiato per poter fornire delle prove (la targa del responsabile ad esempio) in caso di danneggiamento dovuto ad un urto con altri veicoli. Il sistema è, nel suo complesso, formato da due componenti di cui il primo, sopra descritto, è installato a bordo del veicolo ed il secondo è rappresentato da un'applicazione per smartphone che può essere utilizzata dal proprietario per recuperare i dati memorizzati in seguito ad una collisione.

Un esempio del funzionamento del sistema è il seguente: prima di effettuare una sosta o un parcheggio, il proprietario del veicolo decide di attivare il sistema installato a bordo per monitorarlo durante la sua assenza. Una volta ritornato al veicolo, il proprietario nota un danno e non è in grado di ricostruirne la dinamica a causa della mancanza di testimoni o telecamera di sicurezza. Utilizzando l'applicazione installata sullo smartphone, è in grado di collegarsi al sistema e scaricare il video e i dati di accelerazione registrati durante la collisione. Osservando il video, che riprende sia istanti precedenti che successivi all'urto, il proprietario ha così la possibilità di rivedere l'accaduto e ottenere informazioni utili per il riconoscimento del responsabile. Il video può infine essere utilizzato come prova del danno ricevuto contro il proprietario del veicolo che lo ha provocato. I dati possono essere scaricati del proprietario solo tramite l'applicazione smartphone, utilizzando un canale di comunicazione a corto raggio instaurato con il sistema e vengono trasmessi in forma criptata per ragioni di sicurezza. Vale la pena far notare che il proprietario ha il pieno controllo delle informazioni relative al proprio veicolo, in quanto vengono memorizzate solo nel sistema e non trasmesse in rete.

1.5 Architettura del sistema

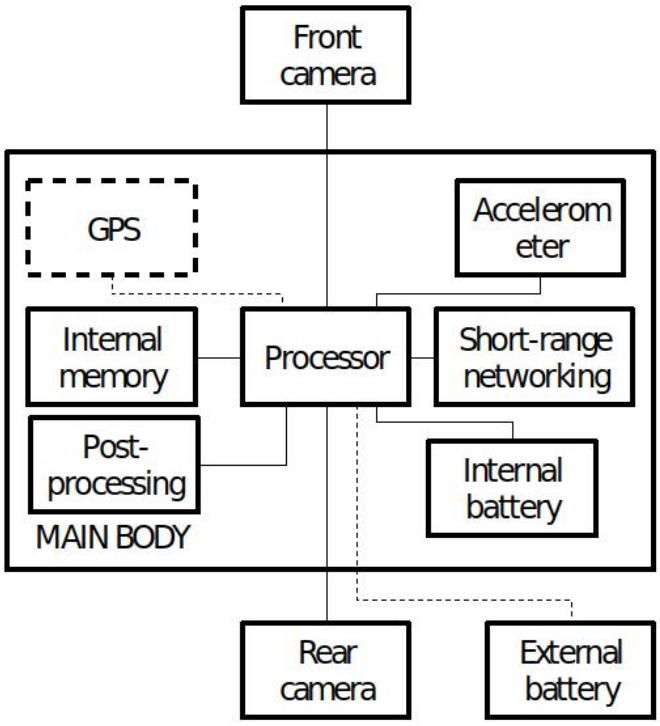

Figura 1.1. Architettura del sistema

La figura 1.1 mostra l'architettura del sistema e i suoi componenti principali da cui è composta:

- Main body: è l'involucro esterno del sistema ed è pensato per essere installato a bordo del veicolo. E' composto da:
  - Processore: il suo scopo è quello di registrare continuamente i frame dalla video camera e i valori di accelerazione per rilevare un'eventuale collisione. Questi dati vengono salvati nella memoria interna e periodicamente sovrascritti in assenza di urti ricevuti dal veicolo. In caso di collisione il processore individua un intervallo temporale composto da alcuni secondi prima e dopo la collisione, recupera i relativi dati della video camera e dall'accelerometro e assegna loro un timestamp per tenere traccia del momento esatto in cui è avvenuta.
  - Memoria interna: memorizza i frame della video camera e i valori di accelerazione rilevati dall'accelerometro.
  - Accelerometro: rileva in maniera continua l'accelerazione del veicolo, calcolata sui 3 assi e viene utilizzata per misurare l'entità della collisione.
  - Modulo di comunicazione a corto raggio: consente la comunicazione con l'applicazione del proprietario quando si trova in prossimità del veicolo. Può essere utilizzato sia per lo scambio di dati che per l'invio di comandi di accensione o spegnimento al sistema.

  - Modulo di post processing: si occupa del processing del video a seguito di una collisione.
  - Batteria interna: fornisce alimentazione al sistema, ma può essere sostituita da altre fonti esterne.
  - Ricevitore GPS: è un modulo opzionale che può fornisce la posizione esatta del veicolo e viene associata agli altri dati rilevati nel momento della collisione.

- Video camere: sono installate a bordo del sistema e consentono il monitoraggio dell'esterno del veicolo durante le fasi di sosta o parcheggio e sono di tipo wide-angle, per garantire una visuale completa. Il numero di video camere da utilizzare è variabile, una possibile configurazione presenta 2 video camere, una frontale e l'altra posteriore.

- Alimentazione esterna: fornisce alimentazione supplementare al sistema e può essere, ad esempio, la batteria del veicolo.

- Applicazione per smartphone: è utilizzata dal proprietario per interagire con il sistema consentendo il download dei video e dei dati di accelerazione: per la comunicazione viene creato un canale sicuro e cifrato per lo scambio dei dati. Notifica all'utente la presenza di nuovi dati nel sistema, relativi ad una collisione avvenuta in sua assenza.

1.5.1 Casi d'uso

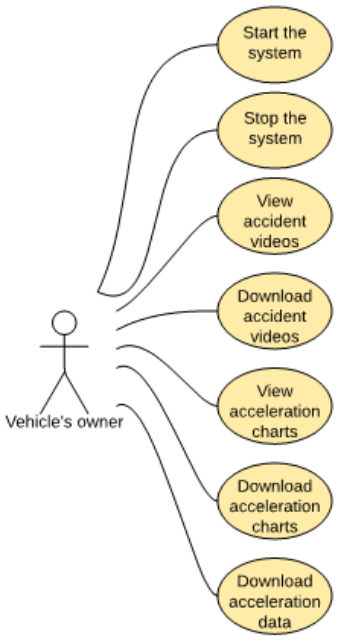

Figura 1.2. Diagramma dei casi d'uso

1.5.2 Diagramma di flusso

In figura 1.3 è riportato il diagramma di flusso che descrive le operazioni svolte dal dispositivo a bordo del veicolo durante. Dopo la sua attivazione, il dispositivo registra continuamente i dati provenienti dall'accelerometro ed i frame dalla video camera. Nell'eventualità di una collisione vengono salvati in memoria sia i frame che i valori di accelerazione e viene generato un alert per comunicare l'evento al proprietario, notificandolo tramite l'applicazione. Come è possibile vedere dal diagramma, dopo un primo urto, il sistema non si spegne, ma ritorna allo stato iniziale, avendo così la possibilità di poter rilevare ulteriori collisioni durante tutto il periodo in cui è in funzionamento.

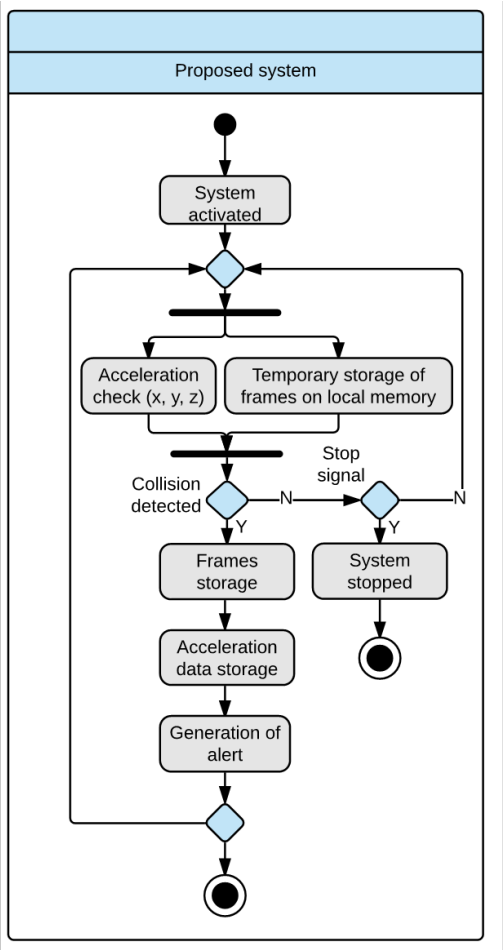

Figura 1.3. Activity diagram: rilevamento di una collisione e salvataggio dei dati

La figura 1.4 mostra, invece, la sequenza di operazioni svolte dal sistema per lo scaricamento del dati in seguito ad una collisione. L'applicazione si collega al dispositivo a bordo dei veicolo e controlla la presenza di nuove collisioni: in caso positivo, l'applicazione inizia il download del video e dei dati di accelerazione rilevati che vengono memorizzati all'interno dello smartphone dell'utente.

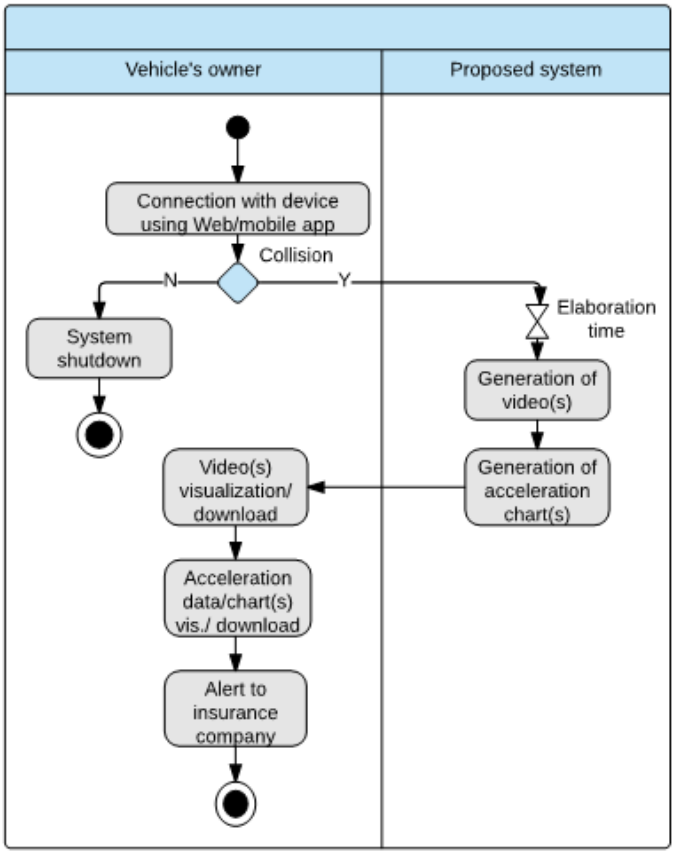

Figura 1.4. Activity diagram: scaricamento dei dati in seguito ad una collisione

Capitolo 2

Framework e hardware utilizzati

In questo capitolo verranno descritti in maniera più dettagliata le componenti hardware utilizzati per realizzare l'architettura del sistema descritto nel capitolo precedente analizzando le loro caratteristiche e le motivazioni che hanno portato alla loro scelta. Inoltre viene fornita una panoramica degli strumenti software usati per l'implementazione e la configurazione di ogni dispositivo utilizzato.

2.1 Descrizione dei componenti

Il processore rappresenta uno dei componenti fondamentali dei sistema, non solo per il suo ruolo di coordinatore, ma anche per le risorse computazionali che mette a disposizione per la ricezione dei dati e la loro elaborazione. Inoltre deve offrire la possibilità di collegamento a tutti gli altri componenti del sistema, accelerometro e video camere in primis, ma anche al modulo opzionale GPS e ad una tecnologia per lo scambio di dati con l'applicazione dell'utente. Le sue caratteristiche hardware devono essere adeguate per lo scopo: un sufficiente quantitativo di memoria per la gestione dei dati ricevuti e per la loro sovrascrittura periodica, oltre alla potenza computazionale necessaria per il processing delle informazioni. Anche i dispositivi di input devono soddisfare alcuni requisiti: i loro dati infatti saranno utilizzati come prova di una collisione ed in quanto tali, devono essere precisi ed affidabili.

L'accelerometro, in quanto strumento responsabile del rilevamento degli urti, deve riportare la lettura dell'accelerazione frequentemente e con una precisione più alta possibile, in modo da riportare in maniera affidabile l'entità dell'urto. La sensibilità dell'accelerometro è un'altra importante caratteristica perchè rappresenta la soglia oltre la quale lo strumento rileva un cambio di accelerazione e consente di poter identificare un numero maggiore di urti, ma potrebbe non sempre essere una vantaggio. Infatti l'accelerometro deve anche essere in grado di distinguere una reale collisione da un altro evento, come può esserlo la chiusura di una portiera del veicolo ed evitare di riportare all'utente un falso positivo.

Le video camere hanno invece il compito di registrare l'accaduto durante l'assenza del proprietario del veicolo, i video che generano devono essere di elevata qualità per consentire il riconoscimento di dettagli che possono portare all'identificazione del colpevole, ma allo stesso tempo devono anche essere inviati all'applicazione

dell'utente e mantenere basse le sue dimensioni potrebbe essere un vantaggio in termini di efficienza e di tempo di trasmissione. Inoltre devono essere di tipo wide angle che sono caratterizzate da una lunghezza focale minore rispetto alle normali lenti fotografiche e questo consente la ripresa di un angolo visivo molto ampio di circa 180°. In questo modo con due video camera è possibile ottenere una visuale completa dell'esterno dei veicolo.

L'applicazione per smartphone è il componente del sistema con cui l'utente ha la possibilità di interagire direttamente. Le sue funzionalità riguardando principalmente l'accesso ai dati memorizzati dal dispositivo a bordo del veicolo. L'utente deve avere la possibilità di visualizzare i video registrati in seguito ad una collisione, così come i valori di accelerazione, indicativi dell'entità dell'urto ricevuto. L'interfaccia grafica deve essere semplice, per poter individuare rapidamente le funzionalità offerte ed evitare di confondere l'utente. Inoltre deve essere efficiente e questo si traduce nella velocità di trasferimento dei dati tra il dispositivo e lo smartphone, in modo da limitare il tempo necessario per lo scaricamento delle informazioni richieste. La sicurezza è un'altra caratteristica del sistema: il canale di trasmissione instaurato tra le due parti deve essere sicuro per garantire l'autenticità ed integrità dei dati. Infine, la gestione delle notifiche consentono all'utente di sapere se ci sono state nuove collisioni, senza dover accedere al dispositivo e controllare manualmente la presenza di un nuovo video.

2.2 Raspberry Pi

Il Raspberry Pi è un single-board computer creato nel 2012 e lanciato dalla Raspberry Pi Foundation, un'organizzazione britannica di beneficenza nata nel 2009 [2]. La scheda è stata realizzata con lo scopo di avvicinare persone di tutte le fasce di età al mondo del computer ed in particolare alla programmazione, infatti Pi sta per Python, un noto linguaggio di programmazione orientato agli oggetti. Il Raspberry Pi è compatibile con numerosi sistemi operativi basati su kernel Linux, ma viene raccomandato l'utilizzo di Raspbian, un sistema operativo derivato da Debian e ottimizzato per il suo hardware. Il modello utilizzato è il 3B+ che si differenza dal predecessore 3B per la qualità dei componenti installati, che garantiscono migliori performance e fluidità, a parità di dimensioni e prezzo.

Il Raspberry è adatto per la creazione di diverse tipologie di applicazioni, sopratutto in ambito IoT (**I**nternet **o**f **T**hings), la cui diffusione ha portato ad un rapido e crescente utilizzo della scheda. Infatti, grazie alla presenza di numerose porte hardware, offre la possibilità di collegare dispositivi USB come mouse, tastiera o schermi, ma anche generici sensori tramite i pin di input/output. Il punto di forza del Raspberry è sicuramente la sua flessibilità di utilizzo e la vasta gamma di applicazioni che è in grado di realizzare: da semplice strumento per avvicinare gli studenti alla programmazione, alla creazione di complessi sistemi embedded. Per quanto riguarda la connettività, il modello 3B+ offre una connessione wireless LAN dual band (2.4 e 5.0 GHz), una comunicazione Bluetooth 4.2 Low Energy e 4 porte Ethernet con velocità limitata a 300 Mbit s, con possibilità di utilizzare la tecnologia PoE (**P**ower **o**ver **E**thernet) per l'alimentazione. Lo slot per micro SD

viene utilizzato sia per il caricamento del sistema operativo Raspbian che per la memorizzazione dei dati.

2.2.1 Architettura Hardware

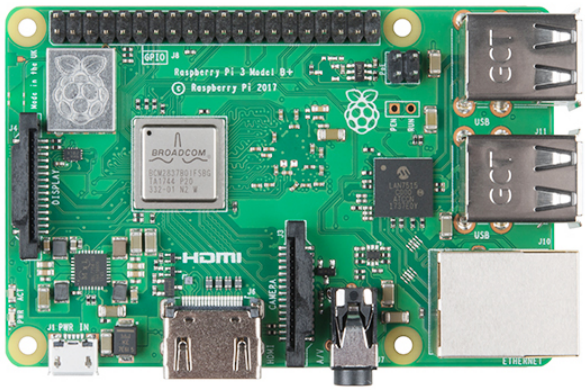

Figura 2.1. Raspberry Pi 3 B+

Il Raspberry Pi Model 3 B+, in figura 2.1, è dotato di un system-on-chip (SoC) Broadcom BCM2837B0, con un processore 1.4 GHz ARM Cortex-A53 (Armv8) quad-core a 64 bit, 1 GB di RAM, 4 porte USB, una porta HDMI e un lettore di schede MicroSD da cui è possibile far partire il sistema operativo. Inotre dispone di 4 porte GBit-Ethernet, Wi-Fi a 5 GHz e Bluetooth V4.2 [3]. Il Raspberry è il punto di partenza del sistema sviluppato, senza il quale non sarebbe stato possibile fornire supporto ai dispositivi collegati, ed è in grado di effettuare la necessaria computazione per l'analisi dei dati ricevuti. Rappresenta il cervello dell'intero progetto ed orchestra contemporaneamente tutti i componenti che ne fanno parte, tramite le diverse interfacce di collegamento.

L'accelerometro è stato installato su una breadboard e collegato al Raspberry per mezzo dei pin I^2C che consentono un facile utilizzo del protocollo di comunicazione. La video camera, invece, viene collegata ad una porta usb sia per l'alimentazione che per l'invio dei singoli frame al Raspberry. La comunicazioni con entrambi i dispositivi collegati è possibile grazie al kernel Linux su cui è basato Raspbian che fornisce supporto nativo per il protocollo I^2C e per i driver UVC con cui è compatibile la video camera.

2.3 Android

Android è attualmente il principale sistema operativo open-source mobile, installato su un elevato numero e varietà di dispositivi. Progettato inizialmente per dispositivi touchscreen, come smartphone e tablet, è stato in seguito adattato per essere eseguito anche su smartTV, laptop ed altri numerosi dispositivi mobile [4]. Android è stato presentato nel 2007 da un gruppo di imprese tecnologiche, di cui Google è il leader, denominato OHA (**O**pen **H**andset **A**lliance) che si occupa della definizione di open standards per dispositivi mobile [5].

L'architettura di Android è formata dal sistema operativo e dalla piattaforma software Android SDK che include numerosi strumenti per lo sviluppo di applicazioni, come debugger, librerie ed un emulatore. Le applicazioni Android sono scritte principalmente in Java, ma sono disponibili anche delle API che consentono l'utilizzo di C++ per lo sviluppo. Android è basato sul sistema operativo Linux, di cui possiede una versione ridotta. Il kernel Linux si occupa dell'interazione con l'hardware sottostante, della gestione della memoria e della sicurezza del sistema operativo.

La sicurezza è uno dei pilastri portarti dell'architettura Android: ogni applicazione viene eseguito all'interno di un proprio processo, su un'istanza di della macchina virtuale Dalvik, simile alla Java Virtual Machine, ma ottimizzata per l'esecuzione su dispositivi mobile. Inoltre ad ogni applicazione viene associato un proprio user ID per poter sfruttare le funzionalità del kernel Linux che consente al sistema operativo di riconoscere le risorse di un'applicazione ed isolarle dagli altri processi, in modo che possano essere protette da applicazioni malevole.

2.3.1 Componenti di Android

Un'applicazione Android può essere composta dai seguenti componenti principali del sistema operativo:

- Activity: componente dotato di interfaccia grafica, che consente l'interazione con l'applicazione. Generalmente un'applicazione è costituita da più activity, una per ogni operazione che si vuole fornire all'utente.

- Service: componente privo di interfaccia grafica, utilizzato generalmente per effettuare lunghe operazioni in background, senza bloccare l'interazione dell'utente con altre applicazioni.

- Content Provider: componente che si occupa della gestione dei dati di un'applicazione e fornisce accesso ad interfacce standard che consentono la condivisione dei dati con altre applicazione, in maniera sicura e controllata dal sistema operativo.

- Broadcast Receiver: componente privo di interfaccia grafica, utilizzato per consentire all'applicazione di essere notificata in seguito ad un certo tipo di evento. Il sistema operativo genera diversi messaggi che possono essere ricevuti dalle applicazioni, come la notifica di batteria scarica o spegnimento dello schermo.

2.3.2 Manifesto

Il manifesto è un file XML, definito per ogni applicazione. Il suo scopo è quello di raccogliere tutte le informazioni che la riguardano, tra cui tutti i componenti di Android che vengono utilizzati. Inoltre, un'applicazione deve dichiarare all'interno del manifesto tutte le risorse del sistema operativo che intende utilizzare, come l'invio di SMS, l'utilizzo della rete e l'accesso al file system. Anche i componenti hardware utilizzati devono essere presenti nel manifesto, come la richiesta di accesso alla fotocamera. In mancanza dei permessi necessari, Android blocca qualsiasi tentativo di accesso a risorse non autorizzate. Dalla versione 6.0 del sistema operativo, i permessi devono anche essere accettati dall'utente per consentire all'applicazione di accedere alle risorse richieste.

2.3.3 Activity

Le activity rappresentano il componente principale delle applicazione perchè forniscono all'utente un'interfaccia grafica con cui interagire. Android gestisce le activity del dispositivo come una pila, definita stack. Un'activity può decidere di inizializzarne un'altra, che viene messa al primo posto dello stack, diventa visibile e abilita l'interazione da parte dell'utente. La precedente activity, invece, rimane in attesa, fino a che la più recente è ancora attiva. Ogni volta che un activity termina autonomamente o dopo la pressione del pulsante indietro, la seconda dello stack prende il suo posto. Le activity dello stack hanno anche una diverse priorità impostata dal sistema operativo: le ultime della pila sono possono essere terminate forzatamente da Android nel caso in cui non ci siano abbastanza risorse per mantenere in memoria le altre activity.

Il compito principale delle activity è l'acquisizione delle risorse richieste dell'applicazione ed il successivo rilascio, come il caricamento di contenuti o l'utilizzo di sensori hardware. Inoltre devono gestire l'interazione dell'utente con i componenti che gli vengono presentati nell'interfaccia grafica. Il ciclo di vita delle activity è un altro concetto fondamentale dell'architettura Android che delimita l'intervallo di tempo in cui sono mantenuta in memoria. Il sistema operativo fornisce le risorse necessarie per l'esecuzione delle activity all'interno dello stack ed invia delle notifiche per comunicare le diverse fasi in cui l'activity si trova. Il ciclo di vita è strettamente legato anche alla visibilità delle activity che dipende dallo stato in cui si trovano. Di seguito le principali callback che Android invia alle activity per comunicare le fasi in cui si trovano; è compito del programmatore reagire a queste notifiche nella maniera più opportuna.

- onCreate: questa funzione può essere chiamata in due differenti scenari. La prima volta che l'activity viene eseguita oppure quando viene rilanciata dopo una sua prematura terminazione, a causa della mancanza di risorse del sistema operativo. In questa fase l'activity non è ancora visibile all'utente. Le operazioni da effettuare in questa funzione riguardando l'inizializzazione delle risorse e la preparazione dell'interfaccia grafica.

- onStart: funziona chiamata quando l'activity diventa visibile all'utente oppure quando lo diventa nuovamente. In quest' ultimo caso, viene preceduta da una chiamata al metodo onRestart.

- onResume: il sistema operativo chiama questa funzione quando l'activity raggiunge il primo posto dello stack e consente all'utente di interagire con essa.

- onPause: l'activity viene spostata al secondo posto dello stack. L'applicazione è in background e devono essere rilasciare tutte le risorse di cui non ha bisogno in questa fase.

- onStop: metodo chiamato quando l'activity non è più visibile all'utente. Può essere seguito da onRestart se l'activity viene spostata nuovamente al primo posto dello stack oppure da onDestroy se sta per essere terminata.

- onDestroy: ultima notifica che il sistema operativo invia all'applicazione che sta per essere rimossa dalla memoria. Può essere invocata in maniera autonoma o Android che necessita di risorse per altre activity.

Creazione dell'interfaccia

All'interno del metodo onCreate l'activity deve creare una view, che estende la classe base android.view.View, ovvero la schermata che verrà mostrata all'utente ed impostarla con il metodo setContentView. Una view è composta da numerosi componenti grafici, definiti widgets, che occupano una specifica posizione sullo schermo del dispositivo. La struttura dei componenti grafici è ad albero: viene definito un componente principale che può contenere al suo interno altri elementi. La creazione dell'interfaccia può seguire diversi approcci:

- Creare la view da codice: consente al programmatore di avere il controllo completo dei componenti grafici ed è semplice modificarne il comportamento e l'aspetto dinamicamente. Lo svantaggio è l'impossibilità di avere un riscontro visivo dell'interfaccia prima dell'esecuzione effettiva dell'applicazione.

- Creazione della view con un file XML: il metodo setContentView può ricevere come parametro un intero, che corrisponde all'identificativo del file XML, in cui è viene descritto sia l'aspetto che il comportamento degli elementi, ad esempio in seguito all'interazione dell'utente. Il vantaggio di questo approccio è semplicità di utilizzo, è possibile visualizzare l'interfaccia grafica mentre viene realizzata tramite l'editor messo a disposizione dall'ambiente di sviluppo. La gestione dinamica dei contenuti, invece, risulta più complessa.

- Utilizzare un approccio ibrido tra i precedenti: permette la creazione dell'albero delle view con il file XML e la personalizzazione dei componenti da codice. Ad ogni elemento dell'interfaccia grafica può essere associato un id univoco, utilizzato per ottenere un riferimento a quell'elemento da codice. Questo approccio consente la separazione della logica degli elementi dal loro rappresentazione sullo schermo.

Oltre alle activity, Android ha aggiunto negli ultimi anni un'altra classe che può essere utilizzata per la realizzazione dell'interfaccia grafica. Le motivazioni riguardano sopratutto la gestione dell'interfaccia, che potrebbe non essere adeguata per tutti i dispositivi. Le diversi dimensioni dello schermo e la differente risoluzione

rendono difficile realizzare un'interfaccia che si adatti facilmente a tutti gli smartphone. La soluzione prevista da Android è la suddivisione di un activity in altri componenti che possono essere combinati insieme per adattarsi meglio allo schermo del dispositivo: i fragment.

Fragment

I fragment sono stati introdotti per semplificare l'adattabilità dell'interfaccia grafica alle diverse dimensioni degli schermi. Un'activity può essere composta da più fragment, ognuno dei quali ha una propria interfaccia grafica che risponde alle interazioni dell'utente. I fragment rendono un'applicazione dinamica, possono essere rimossi o aggiunti durante la sua esecuzione ed essere riutilizzati all'interno di altre activity. Il ciclo di vita dei fragment è strettamente legato a quello dell'activity che li contiene: ogni callback ricevuta dall'activity provoca lo stesso cambiamento di stato anche nei fragment contenuti. Android fornisce alcune notifiche aggiuntive ai fragment:

- onAttach: chiamata quando il fragment viene associato alla rispettiva activity.
- onCreateView: il sistema operativo chiede al fragment di creare la proprio interfaccia grafica.
- onActivityCreated: comunica al fragment che l'activity è stata completamente create, in seguito alla terminazione del suo metodo onCreate.
- onDestroyView: l'interfaccia grafica del fragment deve essere rilasciata.
- onDetach: il fragment sta per essere dissociato dall'activity.

L'activity assume anche il ruolo di coordinatore e consente la comunicazione tra i diversi fragment che ospita. Pratica comune è la definizione di un'interfaccia all'interno dei fragment, che viene implementata dall'activity che li contiene. Questo meccanismo permette ai fragment di notificare il verificarsi di un evento agli altri fragment o all'activity, così come è possibile scambiarsi dei dati.

La descrizione dei rimanenti componenti principali di Android è riportata nelle successive sezioni con lo scopo di fornire anche una panoramica del loro utilizzo all'interno del sistema.

2.4 Qt

La Trolltech, società di software norvegese, iniziò lo sviluppo di Qt nel 1991 ed è stato rilasciato nel 2005. La prima versione della libreria era disponibile solo per Unix e Windows, successivamente è stata creata una versione compatibile anche con MacOs. Qt è una libreria multi piattaforma utilizzata principalmente per la creazione di applicazioni dotate di interfaccia grafica che possono essere eseguite sui principali sistemi operativi (desktop, mobile ed embedded). Grazie al Qt Platform Abstraction, uno strato software che consente l'esecuzione delle applicazioni Qt indipendentemente dalla piattaforma, senza dover cambiare il codice sorgente. Oltre alle interfacce grafiche, è possibile utilizzare le classi offerte da Qt per la gestione dei file, delle comunicazioni di rete, connessioni con database e molte altre

funzionalità, che lo rendono un framework completo per la realizzazione di applicazioni ricche ed efficienti. Qt sfrutta estensivamente il preprocessore C per arricchire principalmente il linguaggio di programmazione C++, ma fornisce interfacce anche per altri linguaggi [6].
Qt Creator è l'IDE adatto per la realizzazione di applicazioni Qt: di base un ambiente di sviluppo C++ ma fornisce una semplice interfaccia di utilizzo per la creazione di elementi grafici Qt. Inoltre si integra perfettamente con numerosi tool interni messi a disposizione da Qt come qmake, uno strumento per la generazione dei makefile necessari per la compilazione di un programma Qt. Come tutti i linguaggi ad oggetti, anche Qt dispone della classe base QOject da cui derivano tutte le altre classi. QObject non rappresenta solo la radice della gerarchia, ma fornisce delle funzionalità aggiuntive, tra cui il parenting system e il costrutto dei signal e slot. Il parenting system consente ad ogni QObject di avere dei parent e dei children, ovvero creare una dipendenza tra oggetti. Questo è molto utile sopratutto durante la distruzione di un oggetto, in cui viene assicurata la distruzione anche dei children, sollevando il programmatore da quell'onere ed evitando memory leaks.

2.4.1 Signals e slots

Specialmente nelle applicazioni grafiche, tutte le librerie forniscono dei meccanismi per rilevare un'azione da parte dell'utente e rispondere a quell'evento. A prescindere dal loro nome o dalla loro implementazione, ciò che li accomuna è il funzionamento basato sul pattern observer che prevede la distinzione in due tipologie di oggetti: observable e observer. Un observer si dichiara interessato ai cambi di stato dell'oggetto observable, il quale si impegna a notificarlo quando ciò accade. Anche in Qt è stato implementato un meccanismo simile, ma piuttosto che avere queste due tipologia di oggetti, viene utilizzato un costrutto di alto livello formato da signals e slots. Un signal è fondamentalmente un messaggio che viene inviato da un certo oggetto per comunicare qualcosa, spesso associato ad un suo cambio di stato. Uno slot invece è una funzione che viene utilizzata per ricevere questo messaggio. Molte classi di Qt implementato di base questo meccanismo, come QPushButton che possiede il signal *clicked* che viene emesso a seguito del relativo evento, ma è possibile crearne di nuovi in base alle proprio esigenze. I signal e slot sono di base delle semplici funzioni che possono essere utilizzate normalmente, ma per consentire la comunicazione tra un oggetto e un altro devono essere collegati, come mostrato nell'esempio.

```
QObject::connect(QObject *sender, SIGNAL(send()),
QObject *receiver, SLOT(receive()))
```

E' possibile usare le macro SIGNAL e SLOT di Qt come sintassi semplificata per la funzione connect, assumendo che il sender abbia un signal send e il receiver un slot chiamato receive: questo punto, il sender può notificare il receiver con emit send(). E' anche possibile inserire dei parametri nei signal e slot per inviare informazioni tra oggetti, piuttosto che limitarsi a notificare un certo evento.

2.4.2 Meta Object Compiler

A supporto di questo costrutto e di altre funzionalità, Qt mette a disposizione un meta-object system in grado di consentire l'utilizzo di particolare paradigmi non supportati nativamente da C++, come l'introspection e la chiamata di funzioni asincrone. Il **M**eta**O**bject **C**ompiler (noto come moc) è uno strumento di Qt che controlla la presenza di meta-information all'interno delle classi, le interpreta e ne rende possibile l'esecuzione. La sintassi dei signal e slot, infatti, non è comprensibile da un normale compilatore C++ e senza una traduzione non sarebbe possibile utilizzarli. La macro Q_OBJECT, specificata nel header file di una classe, comunica al moc la presenza di meta-object code che deve essere tradotto in sintassi regolare del C++.

In conclusione, il costrutto signals/slots rappresenta l'alternativa di Qt alle callback e grazie alla sua semplicità ed efficienza, consente la creazione di applicazioni event-based, capaci di reagire in seguito ad eventi, non solo generati da elementi grafici, ma anche da componenti interni all'applicazione. Infine, vale la pena far notare che per consentire il loro funzionamento, c'è bisogno di una meccanismo per l'attesa dei signal ed il loro invio agli slot collegati, realizzato sempre da Qt con l'utilizzo della classe QCoreApplication che fornisce un event loop per i programmi senza interfaccia grafica. In questo modo il thread principale del programma si mette in uno stato di attesa, gestendo al suo interno la coda dei eventi in arrivo che verranno smaltiti il prima possibile.

2.4.3 Cross-compilazione

La cross-compilazione è una tecnica che, tramite l'utilizzo di un cross-compilatore, consente la compilazione di codice sorgente per creare un file binario in grado di essere eseguito su architetture e sistemi operativi diversi da quello della macchina su cui è stato generato. La cross-compilazione è utilizzata principalmente per la compilazione di applicazione per sistemi embedded su cui non è possibile compilare a causa della mancanza di risorse hardware o software, come l'assenza di un sistema operativo. Qt Creator è molto utilizzato anche in questo ambito e fornisce una semplice interfaccia per consentire la configurazione di un dispositivo embedded. Per abilitare la cross-compilation è sufficiente cambiare il target dell'applicazione nelle impostazioni del progetto ed in fase di compilazione viene utilizzato automaticamente il cross compiler per generare il file binario. Infine è possibile caricarlo direttamente sul dispositivo e poterne seguire l'esecuzione ed eventualmente il debugging direttamente su una console di Qt Creator.

2.4.4 JSON

JSON (**J**ava**S**cript **O**bject **N**otation) è un formato di testo language-independent per lo scambio di dati. La semplicità di utilizzo hanno contribuito alla sua popolarità, infatti attualmente numerosi linguaggi di programmazione hanno la possibilità di serializzare e deserializzare dati in json. I dati possono essere rappresentati come oggetto (array non ordinato di coppie chiave-valore) oppure array ordinato di valori. Un oggetto in json è rappresentato come un insieme di coppie chiave-valore,

separate dalla virgola, all'interno di parentesi graffe. Un array di valori è una sequenza di dati, separati dalla virgola, delimitata da parentesi quadre. Ogni coppia chiave-valore è separata da due punti [7].
I tipi base supportati da json sono:

- Stringhe
- Numeri
- Boolean
- Oggetto
- Array
- null

La possibilità di serializzare oggetti complessi in semplici stringhe, rende json un formato adatto per l'invio di dati in applicazioni client-server perchè in grado di preservare la struttura delle informazioni. E' anche molto utilizzato per realizzare la persistenza dei dati, memorizzando su file di testo ed in maniera facilmente comprensibile, tutte le informazioni in che devono essere preservate. Sono questi i due scenari in cui è stato sfruttato questo formato testuale.

Il Raspberry mantiene in memoria numerose informazioni, non solo riguardanti i video e i dati di accelerazione, che devono essere ripristinate dopo il suo spegnimento. Qt supporta nativamente json, evitando l'utilizzo di librerie aggiuntive per la serializzazione dei dati. Android, invece, ha richiesto l'impiego della libreria Gson [8], sviluppata da Google, che permette la serializzazione/deserializzazione di oggetti Java in stringhe.

2.5 Accelerometro MPU 6050

Il sensore MPU 6050 InvenSense in figura 2.2 integra su un singolo chip un accelerometro MEMS a 3 assi (Micro Electro-Mechanical Systems) e un giroscopio MEMS a 3 assi. Il sensore è in grado di leggere contemporaneamente i valori di tutti gli assi ed è molto accurato in quanto possiede un convertitore AD (Analogico-Digitale) a 16 bits. La comunicazione con il chip avviene tramite il protocollo I^2C (**I**nter **I**ntegrated **C**ircuit) che prevede lo scambio di informazioni sul bus tra un *master*, rappresentato dal RaspberryPi e il chip, che ricopre il ruolo di *slave*.

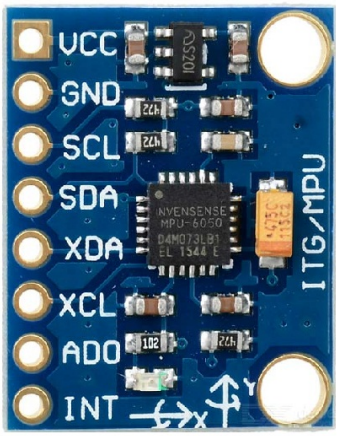

Figura 2.2. MPU 6050

I^2C

E' un protocollo che consente la comunicazione tra circuiti integrati tramite un bus seriale a due fili: SDA (**S**erial **DA**ta) per lo scambio di dati e SCL (**S**erial **CL**ock) per il segnale di clock [9]. Il master è il dispositivo che controlla il segnale di clock, e può inviare o ricevere dati dallo slave, indirizzato dal master su 7 bit. Lo slave, invece, invia o riceve dati dal master, ma non ha alcun controllo sul clock.

Sequenza di trasferimento dati

Ogni lettura o scrittura sul bus deve seguire una precisa sequenza di operazioni:

1. Invio di un bit di START dal master
2. Invio dell'indirizzo dello slave su 7 bit da parte del master
3. Invio di un bit (R/W) per differenziare un'operazione di scrittura da una di lettura
4. Attesa o invio di un bit di ACK
5. Invio o ricezione di un byte di dati
6. Attesa o invio di un bit di ACK
7. Invio del bit di STOP da parte del master

Questa sequenza di riferisce al trasferimento di un singolo byte di dati: i passi 5 e 6 possono essere ripetuti per inviare più byte all'interno della stessa sessione di comunicazione [10].

Segnali di START e STOP

Lo scambio dei dati avviene all'interno di una particolare sequenza di segnali, START e STOP che il master invia sul bus per cominciare o terminare una comunicazione.

- START: il valore di SDA e SCL è logicamente alto, in questa condizione il master imposta il livello di SDA a basso, lasciando alto il segnale di clock, per comunicare allo slave che sta per inviare dei dati sul bus.

- STOP: il valore di SDA e SCL è logicamente basso, il master imposta prima il valore di SCL ad alto ed in seguito anche il segnale SDA, in modo che lo slave capisca che la comunicazione è terminata.

I segnali di START e STOP sono quindi caratterizzati dal cambio di SDA mentre il valore del clock è logicamente alto. Durante l'invio dei dati, invece, il segnale SDA può variare solo mentre il clock è impostato su basso, e il dato inviato o ricevuto viene letto sul fronte di salita o discesa di SCL [11].

Bit di Acknowledge

La ricezione di dati su un bus seriale presuppone sempre l'utilizzo di un meccanismo per confermare il corretto completamento dell'operazione e questo viene realizzato tramite l'invio di un particolare bit, noto come ACK. In seguito ad ogni invio di un byte, il mittente rilascia il bus SDA ed aspetta il successivo periodo di clock per leggere il valore impostato dal ricevente: un valore basso è interpretato come un ACK [12].

Scrittura di un registro

Per inviare dei dati sul bus, il master manda il segnale di START seguito dall'indirizzo dello slave (7 bits) e il bit ($R/W = 0$) che specifica una scrittura in un registro. Dopo aver ricevuto l'ACK, il master invia l'indirizzo del registro (su 8 bits) sul quale vuole scrivere ed aspetta l'ACK da parte dello slave. A questo punto il master procede con l'invio del valore che vuole scrivere all'interno del registro. La sequenza si conclude con l' ACK dallo slave e l'invio dei bit di STOP dal master [13].

Lettura di un registro

La ricezione di dati da parte del master avviene in maniera simile alla scrittura ma necessita di ulteriori operazioni. Come prima il master invia l'indirizzo dello slave con il bit (R/W) impostato a 0 (che significa scrittura) e l'indirizzo del registro. Dopo l'ACK da parte dello slave, il master invia un segnale di START seguito dall'indirizzo dello slave, ma questa volta con il bit $R/W = 1$ per la lettura. A questo punto, il master rilascia la linea dati SDA per consentire allo slave di prenderne il controllo. Per terminare la lettura, il master invia un segnale di NACK (valore logicamente alto di SDA, opposto all'ACK) allo slave per fargli rilasciare il bus dati ed infine invia un segnale di STOP [14].

Repeated START

Il segnale repeated START si differenza dal segnale di START in quanto può essere inviato prima di un segnale di STOP. E' utilizzato per inviare più sequenze di dati all'interno della stessa sessione di comunicazione, senza prima rilasciare il bus con il segnale di STOP e acquisirlo nuovamente con uno di START [15].

Modalità di funzionamento

I dati dall'accelerometro possono essere letti in maniera sincrona o asincrona. Nel primo caso il *master* richiede esplicitamente i dati allo *slave*, tramite il bus. La seconda modalità di lettura dei dati prevede l'utilizzo di un buffer FIFO di 1024 byte all'interno del sensore e di un segnale di *interrupt* per comunicare con il *master*: l'accelerometro può essere programmato per salvare i dati rilevati dagli assi in maniera sincrona all'interno del buffer. L'*interrupt* può essere utilizzare per segnalare la presenza di nuovi dati o l'occorrenza di un evento. Per questo lavoro di tesi si è preferita la modalità sincrona, memorizzando i valori letti dal sensore in un buffer.

2.6 Videocamera ELP-USBFHD04H-L180

Figura 2.3. ELP-USBFHD04H-L180

La videocamera è sicuramente uno dei componenti fondamentali del sistema, in quanto deve essere in grado di fornire una chiara rappresentazione dell'accaduto, sotto forma di video, che potrà essere eventualmente utilizzata come prova legale. Questo è il suo scopo principale e prima della scelta definitiva, sono stati selezionati e valutati diversi requisiti hardware che doveva possedere la video camera per essere adatta a questo progetto di tesi. In primo luogo, ogni frame deve essere ad alta risoluzione per poter catturare al meglio ogni dettaglio all'interno del video, come ad esempio la targa dei veicolo che ha causato la collisione. Inoltre la frequenza di cattura dei frame deve essere sufficientemente alta per garantire un'adeguata fluidità del video ed evitare che gli istanti salienti non vengano ripresi a causa di un intervallo di tempo troppo elevato tra un frame ed il successivo. Un'altra caratteristica che deve avere la camera è un elevato angolo di campo, ovvero l'estensione

angolare catturata dal fotogramma, così che una singola video camera possa coprire un intero lato dei veicolo senza punti ciechi. In questo modo con due video camere poste nella parte anteriore e posteriore, è possibile ottenere una visuale completa dell'esterno.

In figura 2.3 viene mostrato il modello di video camera utilizzato. La board camera ELP-USBFHD04H-L180 è una video camera USB ideale per numerose applicazioni grazie alle sue caratteristiche tecniche. Le sue ridotte dimensioni e la sua efficienza la rendono adatta sopratutto per la creazione di sistemi di sicurezza e monitoraggio ambientale. Il collegamento con il Raspberry avviene tramite l'interfaccia USB 2.0 di cui è dotata la video camera che è inoltre compatibile con i driver UVC di Linux, che definiscono le funzionalità di streaming dei dispositivi USB.

Modello	ELP-USBFHD04H-L180
Sensore	AR0330
Formato di compressione	H.264/ MJPEG / YUV2
Risoluzione e frame rate	1920 x 1080 H.264 30fps 1280 x 720 H.264 30fps 640 x 480 H.264 30fps 640 x 360 H.264 30fps 320 x 240 H.264 30fps 320 x 180 H.264 30fps

Tabella 2.1. Specifiche tecniche

Nella tabella 2.1 [16] sono state riportate tutte le possibili risoluzioni supportate, ma solo il frame rate relativo al formato di compressione H.264 perchè è quello effettivamente utilizzato. La possibilità di generare frame compressi è stato il criterio principale per la scelta di questo modello di video camera.

H.264

H.264 è un formato di codifica video nato con lo scopo di fornire una tecnologia di compressione capace di ottenere video di alta qualità con un utilizzo inferiore del bitrate. Come immediata conseguenza, uno streaming di dati codificato in H.264 necessita di una larghezza di banda inferiore, senza però incidere sulla qualità del video. Queste sono state le caratteristiche che l'hanno reso particolarmente adatto per le applicazioni di video sorveglianza in cui la qualità dell'immagine è un requisito fondamentale, ma la larghezza di banda è spesso limitata e deve essere sfruttata al meglio. Inoltre i video codificati in H.264 occupano un quantitativo molto minore di memoria in modo da facilitare la loro trasmissione in rete. Per la compressione video in generale vengono utilizzati 3 tipi diversi di frame [17]:

- I-frame: un intra-coded-frame è utilizzato per descrivere un'immagine completa e pertanto non è compressa. Questo tipo di frame non ha bisogno di altri frame per essere decodificato, ma occupa più memoria.

- P-frame: un predicted-frame è espresso in funzione di altri frame e contiene le variazioni nell'immagine rispetto al frame precedente. Ad esempio, riprendendo un paesaggio fisso con un oggetto in movimento, un P-frame necessita solo delle variazioni dell'oggetto, senza riportare i pixel relativi allo sfondo, che è rimasto invariato. La sua occupazione di memoria è ovviamente minore rispetto a quella di un frame di tipo I e questo migliora il rapporto di compressione.
- B-frame: un bi-predictive-picture occupa ancora meno memoria dei frame di tipo P, contenendo le differenze rispetto sia al frame precedente che a quello successivo.

Capitolo 3

Progettazione ed implementazione

3.1 Implementazione

In questa sezione viene trattato il lavoro svolto all'interno della tesi, analizzando le scelte implementative e le motivazioni. Da un punto di vista concettuale, il ruolo centrale del sistema è ricoperto dal Raspberry, il cervello dell'intero dispositivo, responsabile della ricezione dei dati e della coordinazione degli altri componenti utilizzati. Infatti sia l'accelerometro che la video camera funzionano in maniera indipendente l'uno dall'altra, ma devono cooperare per lo scopo finale: fornire i dati relativi alle collisioni rilevate. Inoltre, il Raspberry deve fornire un meccanismo che consenta all'applicazione di ricevere le informazioni che sono memorizzate al suo interno, sfruttando le tecnologie di connessione di cui dispone. Nella prima parte viene analizzata l'interazione con i singoli componenti, descrivendo sia le operazioni necessarie per loro configurazione che per lettura dei dati. In seguito verrà fornita una panoramica del funzionamento generale del sistema mostrando come le informazioni dell'accelerometro e della video camera vengono combinati per la creazione di dati pronti per l'utente. Nella seconda parte viene presentata l'applicazione Android e dell'approccio utilizzato per l'implementazione delle funzionalità offerte all'utente, come la possibilità di scaricare e visionare i video di interesse.

3.1.1 Cablaggio

Il primo passo per la realizzazione del sistema è collegare tra di loro i diversi componenti, in base alle interfacce che mettono a disposizione. La video camera possiede un'interfaccia USB e viene collegata direttamente ad una delle porte del Raspberry. L'accelerometro, invece, non avendo alcune interfaccia, è stato installato su una breadboard e collegato ai pin del Raspberry, come mostrato in figura 3.1[1].

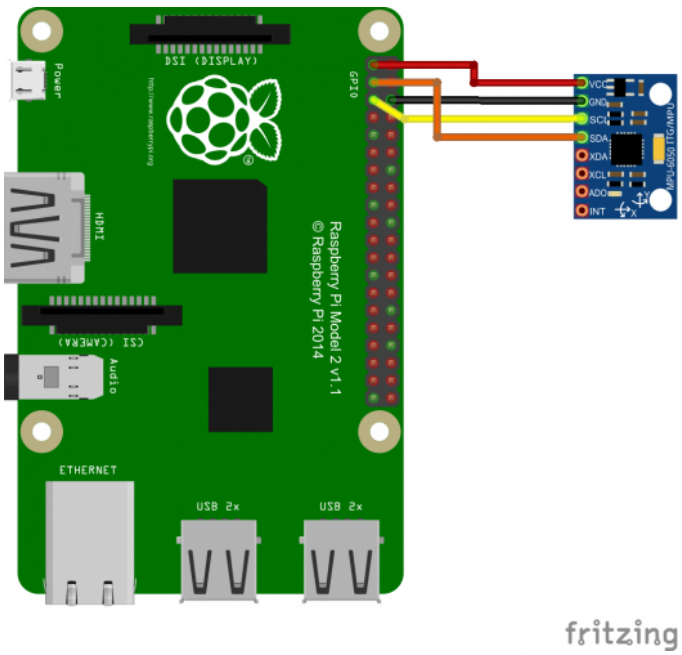

Figura 3.1. Schema di connessione

Il sensore MPU 6050 dispone di 8 pin, ma per il collegamento con il Raspberry sono necessari solo i 4 superiori, riportati nella tabella 3.1. Infatti oltre all'alimentazione e alla massa, sono stati collegati i pin SDA e SCL per consentire la comunicazione sul bus I^2C. I pin XDA e XCL possono essere utilizzati per collegare altri dispositivi di tipo I^2C al sensore, mentre il pin AD0 consente la connessione di un altro sensore di tipo MPU 6050 sullo stesso bus, fornendo un secondo indirizzo. Infine il pin INT abilita la modalità interrupt, con la quale si può ricevere un segnale dal sensore quando sono disponibili nuovi dati da leggere.

Raspberry	**MPU 6050**
Pin 1 (3.3V)	VCC
Pin 3 (SDA)	SDA
Pin 5 (SCL)	SCL
Pin 6 (GND)	GND

Tabella 3.1. Collegamenti

[1]Lo schema di collegamento è stato realizzato con il software Fritzing [18]

3.1.2 Registri

Come precedentemente descritto, lo scambio di informazioni tramite il protocollo I^2C avviene tramite la lettura e la scrittura dei registri interni al sensore, ognuno identificato da uno specifico indirizzo che il master deve conoscere per poter istruire lo slave sulle operazioni da effettuare. L'elenco completo dei registri dell'accelerometro MPU 6050 è riportato sulla relativa register map [19] che fornisce una descrizione generale e l'indirizzo di ognuno. Nello specifico sono stati utilizzati solo alcuni dei possibili registri, tra cui quelli necessari per l'iniziale configurazione e i registri contenenti i valori di accelerazione rilevati.

Indirizzo	**Registro**
0x6B	PWR_MGMT_1
0x3B	ACCEL_XOUT_H
0x3C	ACCEL_XOUT_L
0x3D	ACCEL_YOUT_H
0x3E	ACCEL_YOUT_L
0x3F	ACCEL_ZOUT_H
0x40	ACCEL_ZOUT_L
0x1C	ACCEL_CONFIG
0x06	XG_OFFS_USRH
0x07	XG_OFFS_USRL
0x08	YG_OFFS_USRH
0x09	YG_OFFS_USRL
0x0A	ZG_OFFS_USRH
0x0B	ZG_OFFS_USRL
0x75	WHO_AM_I

Tabella 3.2. Principali registri

3.2 Configurazione dell'accelerometro

Il collegamento dell'accelerometro al Raspberry è relativamente semplice, ma è comunque possibile commettere degli errori in fase di cablaggio. E' infatti sempre buona norma verificarne la correttezza per essere sicuri di star comunicando con il giusto dispositivo. il sensore mette a disposizione il registro WHO_AM_I che contiene un valore di default che identifica l'indirizzo dell'accelerometro sul bus, per cui è sufficiente leggerlo e confrontarlo con il valore atteso: 0x68 nel caso del sensore MPU 6050.

Secondo il datasheet del sensore [21], dopo l'accensione l'accelerometro di trova in una specie di "sleep-mode" che deve essere disabilita per iniziare la lettura dei dati. A questo scopo è possibile azzerare il registro PWR_MGMT_1 per risvegliare il sensore e avviare il calcolo dell'accelerazione. Prima di iniziare la lettura

dei registri, sono necessarie ulteriori configurazioni: l'impostazione della sensibilità dello strumento, che supporta diversi fondo scala e la sua calibrazione [20].

Scelta del fondo scala

Il datasheet [21] riporta la sensibilità dello strumento per ogni fondo scala, come mostrato nella seguente tabella.

Fondo scala	Sensibilità	Unità di misura
$\pm$ 2g	16,384	LSB/g
$\pm$ 4g	8,192	LSB/g
$\pm$ 8g	4,096	LSB/g
$\pm$ 16g	2,048	LSB/g

Tabella 3.3. Sensibilità

I valori di accelerazione restituiti dal sensore sono dati grezzi e senza un'appropriata conversione sono privi di significato o comunque non comprensibili se non si conosce il fondo scala utilizzato. In base al fondo scala scelto, è possibile ottenere dalla tabella il rispettivo fattore di conversione, che permette di esprimere i dati grezzi in funzione dell'accelerazione gravitazionale. Il valore di sensibilità viene calcolato sulla base di tutti i possibili 2^{16} valori che il sensore è in grado di restituire, perchè ogni registro di accelerazione è di 16 bit, diviso l'ampiezza del fondo scala scelto. Prendendo come esempio il primo fondo scala $\pm 2g$, che ha un range di 4g, il valore di sensibilità è 2^{16} / 4 = 16,384. L'unità di misura LSB/g significa che ogni variazione del bit meno significativo del valore di accelerazione, corrisponde ad una variazione di 1/16.384 espressa in funzione dell'accelerazione gravitazionale. Il fondo scala può essere impostato utilizzando il registro ACCEL_CONFIG, scegliendo tra i 4 range supportati dal sensore.

Calibrazione

Infine bisogna calibrare il sensore a causa dello zero-error, ovvero una misurazione errata dell'accelerazione che riporta valori diversi da zero anche in condizioni stazionarie. Per la calibrazione sono disponibili due registri per ogni asse di misurazione, riportati nella tabella 3.2. Il valore di questi registri deve essere sovrascritto con il valore dei rispettivi offset, necessari per rimuovere lo zero-error e quindi ottenere misure più precise. Gli offset sono stati calcolati tramite un processo iterativo che calcola la media di un elevato numero di misurazioni effettuate sui 3 assi, ogni volta con un valore di offset minore del precedente. Questo procedimento termina quando il valore di accelerazione sugli assi x e y è circa zero e vicino ad 1.0 nel caso dell'asse z, per effetto dell'accelerazione gravitazionale.

3.2.1 Lettura dei dati dal sensore

Dopo aver effettuato le configurazioni preliminari, si può procedere con la lettura dei dati dall'accelerometro. Per poter comunicare con un periferico I^2C è necessario ottenere un riferimento al bus su cui si vuole comunicare e sul Raspberry è possibile farlo tramite la system call *open* che restituisce il file descriptor relativo per consentire l'utilizzo delle funzioni *read* e *write*. Secondo il protocollo di comunicazione, ogni registro del sensore viene indirizzato su 7 bit a cui segue un bit aggiuntivo per differenziare le operazioni di lettura e scrittura, impostato automaticamente dopo l'utilizzo della system call *ioctl* che specifica il ruolo di slave del sensore.

Nel seguente esempio viene mostrata una lettura dell'accelerazione relativa all'asse x. E' possibile notare un'operazione di scrittura prima di effettuare una lettura, questo perchè il master deve segnalare il registro a cui vuole accedere, prima di leggerne il contenuto. In particolare, nella funzione read vengono specificati 2 byte da leggere ed è dovuto al fatto che il sensore riporta i valori di accelerazione su 16 bi e vengono riportati su 2 registri da 8 bit con indirizzi contigui. Le possibilità di lettura sono quindi due: la prima modalità prevede la lettura separata dei registri ACCEL_XOUT_H e ACCEL_XOUT_L di 8 bit che contengono rispettivamente i bit più e meno significativi dell'accelerazione totale. Oppure è consentito leggere direttamente 16 bit a partire da ACCEL_XOUT_H, come da esempio.

```
unsigned char buffer[2];
unsigned char register = ACCEL_XOUT_H;
write(fd,register,1);
read(fd,buffer,2);
int16_t accel_x = (buffer[0] « 8) + buffer[1];

float g_force_conversion = 1/ 16.384 ;
float accel_xg = accel_x * g_force_conversion;
```

Infine viene calcolata l'accelerazione totale per avere il valore complessivo dello spostamento del veicolo relativo a tutti gli assi. In realtà, l'informazione più rilevante in questo contesto non è conoscere il valore totale, bensì le variazioni di accelerazione tra un momento ed il successivo che evidenziano maggiormente un brusco movimento da parte del veicolo, rispetto all'istante precedente.

```
float current_xg = accel_xg - previous_xg;
```

float acceleration = $\sqrt{current\_xg^2 + current\_yg^2 + current\_zg^2}$;

L' accelerazione totale viene confrontata con un valore di soglia ad ogni ciclo di lettura per determinare se è sufficientemente alta da essere considerata una collisione. Il valore di soglia determina la sensibilità del sistema intero ed è ciò che discrimina un incidente da un altro tipo di evento, come una portiera chiusa. E' quindi opportuno che sia stabilito in maniera tale non che generi falsi positivi, ma allo stesso tempo deve garantire che anche i minimi urti vengano rilevati.

3.2.2 Memorizzazione dei valori di accelerazione

Come precedentemente detto, il sensore può funzionare sia in modalità asincrona che sincrona. La prima prevedere l'utilizzo del buffer FIFO interno al dispositivo e di alcuni registri che tengono traccia del tipo di informazioni memorizzate e il numero di bytes validi. Un interrupt può essere utilizzato per comunicare la presenza di nuovi dati oppure altri tipi di eventi da parte del sensore. Si è preferita la modalità sincrona perchè era necessario poter accedere non solo ai dati nuovi del sensore, ma anche a valori di accelerazione memorizzati precedentemente.

Buffer circolare

Il sistema deve garantire una lettura continua e costante dei valori di accelerazione, ma deve memorizzare solo quelli relativi ad una collisione ed in particolare devono riguardare sia gli istanti precedenti che i successivi. Per garantire questo meccanismo, i valori di accelerazione letti non vengono subito salvati sul disco del Raspberry, ma mantenuti temporaneamente in una struttura dati in memoria. Solo in caso di collisione viene effettuata una copia parziale di questa struttura dati che viene salvata definitivamente all'interno del Raspberry.

La struttura dati che più si presta a questo scopo è un buffer circolare che, per sua natura, privilegia i nuovi dati a discapito dei più vecchi, che vengono periodicamente sovrascritti. Grazie a questa caratteristica è anche possibile definire una dimensione massima del buffer, in modo che possa occupare un quantitativo costante e limitato di memoria, senza alterando il funzionamento. E' infatti sufficiente scegliere come dimensione massima, un numero poco superiore a quello dei valori che si intende memorizzare contemporaneamente.

La lettura dei dati ed il seguente salvataggio all'interno del buffer sono operazioni che devono essere eseguite con una frequenza abbastanza elevata per poter garantire la corretta individuazione di un urto. Una frequenza di lettura bassa potrebbe provocare la perdita di informazioni, sopratutto in caso di collisione ad alta velocità che provoca un rapido e difficilmente individuabile cambio di accelerazione. D'altra parte, leggere l'accelerazione ad intervalli molto frequenti porta ad una quantità enorme di dati da memorizzare. La scelta della frequenza di lettura deve essere un giusto compromesso tra la facilità con cui il sensore può rilevare un urto e la quantità di informazioni memorizzate per ogni secondo. Sono state provate diverse frequenza tra cui la più soddisfacente, in termini di efficienza, risulta essere 1/3 di secondo.

Per la ricezione dei dati dall'accelerometro è stato utilizzato un thread secondario. Questa scelta è stata fatta sulla base del funzionamento generale di Qt, che per poter gestire correttamente gli eventi generati all'interno del sistema, ha bisogno che il thread principale si occupi di smaltirli all'interno dell'event loop. Il thread si occupa dell'inserimento dei valori nella corretta posizione del buffer, che viene memorizzata nella variabile *currentIndex*. La sovrascrittura degli elementi meno recenti è realizzata proprio grazie a currentIndex, su cui viene effettuata l'operazione modulo al seguito di ogni inserimento. Infatti questo permette il salvataggio dei valori in un intervallo limitato e periodico compreso tra 0 e il numero massimo di elementi - 1.

3.2.3 Rilevare un urto

L'accelerometro misura dunque in ogni istante, durante la fase di parcheggio, il valore di accelerazione del veicolo. In condizioni stazionarie la somma sui 3 assi riporta un valore di circa 1g, dovuta all'asse z su cui agisce l'accelerazione gravitazionale, mentre i restanti assi restituiscono un valore pressochè nullo. Il thread effettua quindi le operazioni di lettura dai dall'accelerometro, salvataggio all'interno del buffer circolare e controllo del valore di soglia. Il caso di interesse è ovviamente quello in cui l'accelerazione è sufficientemente elevata da essere considerata un urto: questa condizione scatena una sequenza di operazioni riguardanti l'intero sistema. La prima procedura da effettuare è il salvataggio dei dati di accelerazione precedenti e successivi all'istante dell'urto, che vengono copiati in un vettore temporaneo.

Copia dei dati di accelerazione

Utilizzare lo stesso thread per realizzare la copia potrebbe portare ad una lettura non attendibile dei valori dall'accelerometro, in quanto avrebbe impiegato del tempo per recuperare i dati precedenti, ritardando la lettura dei nuovi. Per questo motivo viene creato un thread che ha il solo compito di copiare alcuni dati dalla struttura principale. Vengono definite due variabili che delimitano l'intervallo dei dati che verranno copiati all'interno del vettore temporaneo. Ipotizzando di voler memorizzare contemporaneamente i dati di accelerazione relativi a 100 secondi, ad una velocità di lettura di 0.1 secondi dal sensore, avremo una struttura circolare di 1000 elementi. In caso di urto potremmo essere interessati a ricevere informazioni relative ai 30 secondi precedenti e successivi rispetto all'istante dell'urto: con queste ipotesi, necessitiamo una struttura temporanea di 180 elementi.

Le variabili startIndex e endIndex vengono quindi inizializzate al valore rispettivo di currentIndex - 90 e currentIndex + 90, con currentIndex viene rappresentata la posizione del valore di accelerazione superiore alla soglia. Il thread (*workerThread*) deve quindi effettuare una copia dei dati compresi tra startIndex ed endIndex; sebbene la prima metà dei dati sia facilmente recuperabile, non è così per la seconda. Infatti i dati compresi tra startIndex ed currentIndex si riferiscono a valori di accelerazione precedenti all'urto (dovuto al buffer circolare) e quindi devono essere ancora letti dal sensore e salvati in memoria. Dopo aver copiato la prima parte, il workerThread deve infatti attendere che i nuovi dati real-time siano pronti prima di poterli copiare. Questa problematica è stata risolta con l'utilizzo di un costrutto di sincronizzazione.

Condition variable

La classe C++ condition_variable è una primitiva di sincronizzazione, utilizzata per sospendere l'esecuzione di uno o più thread, in attesa che si verifichi una di queste condizioni [22]:

- notifica da parte di un altro thread.
- scadenza di un timeout.
- risveglio spurio.

L'attesa su una condition variable è vincolata dall'acquisizione di un oggetto di tipo unique_lock che, tramite l'utilizzo di un mutex, consente l'accesso ad una risorsa condivisa in mutua esclusione. La classe condition_variable fornisce le funzioni wait e notify: la prima sospende l'esecuzione di un thread mettendolo in uno stato di attesa, mentre la seconda notifica un thread in attesa per consentirgli di procedere nell'esecuzione. La funzione wait accetta come parametro uno unique_lock e una condizione di tipo booleana che se non verificata, lascia il thread in attesa. Un thread che notifica risveglia uno dei thread in attesa sulla condition_variable. La condizione booleana è usata come rimedio al risveglio spurio che è una situazione di errore in cui un thread viene notificato nonostante la condizione non si sia ancora verificata.

Nel caso in esame, la condition_variable mette in attesa il workerThread fino a che l'altro non ha letto e memorizzato un nuovo dato dall'accelerometro. Con questa soluzione è possibile notare una veloce copia dei dati nell'intervallo che va da startIndex a currentIndex, poichè i dati sono precedenti all'urto. Mentre nella seconda metà dell'intervallo totale, i 2 thread accedono in maniera alternata al buffer circolare: il workerThread ha copiato e quindi incrementato startIndex fino al valore di currentIndex e aspetta che il nuovo dato sia pronto, il primo thread riceve un nuovo dato, lo memorizza, incrementa currentIndex e notifica il workerThread che può quindi accedere al nuovo dato e si rimette in attesa. Questa lettura alternata continua fino a quando il valore di startIndex è uguale a quello di endIndex, che determina la fine dell'intervallo dei dati da copiare. L'acquisizione del mutex, prima dell'accesso agli indici, garantisce la lettura o modifica del loro valore in mutua esclusione, assicurandone la coerenza.

Urti successivi

E' stato considerato anche il caso di più urti successivi in un lasso di tempo molto breve, precisamente prima che il workerThread abbia finito di copiare i dati del primo video. Questa possibilità è stata gestita con una modifica degli indici: endIndex viene ricalcolato nello stesso modo, ma partendo da un diverso valore di currentIndex, evitando che superi il valore di startIndex, controllo doveroso a causa della struttura circolare. Il rilevamento dell'urto non interessa solo la classe che si occupa della memorizzazione dei dati dal sensore, ma anche il modulo che gestisce la lettura dei frame della video camera. Per questo motivo il workerThread effettua un'ulteriore operazione che è l'emissione di un signal, collegato ad uno slot appartenente alla classe che gestisce l'acquisizione da video camera, in modo da effettuare una copia dei frame memorizzati prima e dopo l'urto, in maniera simile a quella appena descritta.

Alla fine del processo di copia il sistema ha a disposizione tutti i valori di accelerazione rilevati dal sensore in un intervallo temporale che comprende l'istante di collisione. Questa rappresenta solo la prima delle informazioni che il sistema utilizza per riproporre all'utente la dinamica dell'accaduto. Nella successiva sezione viene trattata nel dettaglio la video camera e di come i frame che fornisce vengono utilizzati per la creazione del video.

3.3 Configurazione della video camera

La compatibilità della video camera con Linux con UVC, ha reso possibile l'utilizzo di Video4Linux2 (v4l2), un API per Linux che consente la cattura di immagini e la registrazione audio/video da devices USB. Per manipolare un device USB è necessario ottenere il file descriptor che Linux gli ha associato utilizzando la system call *int open(const char *pathname, int flags);*, che apre il file specificato nel parametro *pathname*. Nel caso di dispositivi video collegati ad un sistema operativo Linux, il pathname è solitamente /dev/video*, dove * è un intero che corrisponde ad uno specifico dispositivo collegato. Come prima operazione si è ritenuto opportuno effettuare un controllo sui devices USB collegati al Raspberry, sia per assicurarsi che siano dispositivi in grado di generare immagini, sia per garantire la compatibilità con Video4Linux2. Dopo aver memorizzato la lista dei dispositivi USB collegati, vengono interrogate alcune caratteristiche del device, utilizzando la system call ioctl e alcuni flag specifici della libreria v4l2:

- VIDIOC_QUERYCAP: tutti i device v4l2 supportano questa libreria, è utilizzata per scartare i dispositivi non compatibili.
- VIDIOC_G_FMT: consente di interrogare il formato delle immagini generato dalla video camera collegata ed è utilizzata per scartare i dispositivi che non hanno una specifica risoluzione o non supportano il formato H264.

Con questa interrogazione dei dispositivi video è inoltre possibile supportare un numero variabile e non noto a priori di video camere collegate al Raspberry, senza ulteriori interventi, garantendo scalabilità al sistema.

FFmpeg

Fast **F**orward mpeg è una suite software che comprende numerose librerie e programmi per la gestione di flussi audio e video [23]. Sfrutta le API di v4l2 per l'individuazione e la cattura di video da devices compatibili.

- *libavcodec*: libreria utilizzata per la codifica e decodifica di stream audio e video, contiene l'implementazione di numerosi encoder e decoder dei principali formati.
- *libavformat*: libreria utilizzata per la gestione dei container formats. Si occupa principalmente di effettuare le operazioni di muxing e demuxing.*

I principali tool offerti da FFmpeg sono:

- *ffmpeg*: è un tool utilizzabile da linea di comando che consente la conversione audio/video in diversi formati.
- *ffplay*: è un generico media player, consente la riproduzione video da diverse fonti, comprese video camera USB.

* demultiplexing: processo che suddivide un file multimediale nelle sue componenti, ad esempio audio e video.
multiplexing: processo inverso al demultiplexing.

- *ffprobe*: è uno strumento utile per analizzare file multimedia, mostra informazioni relativa ai diversi stream che lo compongono, in un formato human-readable.

Inizializzazione della videocamera

La classe CameraRecorder utilizza le funzioni messo a disposizione da FFmpeg per le principali operazioni da effettuare con la video camera: inizializzazione, cattura dei frame e creazione di file video in formato mp4. La struttura dati *AVInputFormat* contiene le informazioni relative al formato in input del dispositivo, "video4linux2" nel caso in esame. La funzione *avformat_open_input* è usata per accedere alla video camera e riceve come parametri: il percorso della camera ("/dev/video*"), un AVInputFormat e una struttura dati (AVDictionary) contenente parametri opzionali di cattura, come il numero di fps (fotogrammi per secondo) o la risoluzione desiderata. Come output, la precedente funzione riempe una struttura di tipo AVFormatContext, necessaria per le operazioni di lettura dei frame. Una seconda struttura dati, di tipo AVFormatContext, è stata allocata e riempita per consentire la creazione dei video in formato mp4. aggiungere informazioni sui codec.

Le funzioni av_read_frame e av_write_frame sono utilizzate rispettivamente per leggere un frame dalla video camera e scrivere un frame in coda al video mp4. av_read_frame memorizza i dati di un singolo frame all'interno di una struttura dati di tipo AVPacket, contenente i dati compressi generati dalla camera.

3.3.1 Struttura dati

La caratteristica del sistema di memorizzare i dati dalla video camera in maniera continua, consentendo la sovrascrittura periodica, ha richiesto la creazione di una struttura dati ad hoc per l'inserimento ed il recupero dei frame in fase di creazione del video. Come descritto nella sezione 2.6, ogni secondo di video generato dalla video camera è formato dalla presenza di un primo frame di tipo I seguito da 29 predicted frames. In base a questa caratteristica, è stato pensato di rappresentare la struttura dati come una sequenza di una seconda struttura dati formata da 30 elementi, esattamente i 30 frame utilizzati per descrivere un secondo di video. Così come per l'accelerometro, anche in questo caso la lettura continua dalla video camera è stata delegata ad un thread diverso dal principale.

Memorizzazione dei frame

La fase di inserimento dei frame risulta un po' elaborata, perchè un nuovo frame deve essere inserito nella prima struttura che corrisponde ad un certo secondo di registrazione e poi deve essere collocato all'interno della seconda lista che lo rappresenta. Al contrario, la fase di creazione video è molto più semplice ed immediata: è sufficiente accedere alla struttura esterna per avere a disposizione, per ogni suo elemento, la lista dei 30 frame. L'esempio mostra come viene effettuato l'inserimento all'interno della seconda struttura: *frame_number* è una costante di valore 30. L'utilizzo dell'operatore modulo simula l'inserimento circolare all'interno della struttura, limitando il valore di *index* in un intervallo compreso tra 0 e 29.

```
array<AVPacket, frame_number> buffer;
void frames::push(AVPacket pkt){
   buffer[index] = pkt;
   ++index %= frame_number;
}
void frames::getIndex(){ return index;}
```

L'inserimento nella struttura esterno è realizzato in maniera del tutto simile al precedente, con la differenza che il frame deve essere indicizzato due volte: un primo indice individua l'elemento della struttura esterna, contenente la sequenze dei frame di quel secondo mentre il secondo indice localizza la posizione esatta di quel frame nella lista.

```
array<frames,buffer_size> circularBuffer;
void circularBuffer::push(AVPacket pkt){
   if(circularBuffer[currentIndex].getIndex() == frame_number)
      ++currentIndex %= buffer_size;
   circularBuffer[currentIndex].push(pkt);
}
```

La dimensione di *circularBuffer* è *buffer_size* che definisce il numero massimo di secondi che vengono ripresi e mantenuti all'interno della struttura: l'inserimento di un numero superiore di elementi provoca la sovrascrittura di quelli meno recenti. Anche in questo caso è stato usato l'operatore modulo per incrementare *currentIndex* che tiene traccia dell'elemento corrente in cui vengono inseriti i frame. Da notare il controllo sul numero degli elementi all'interno della lista indicizzata da *currentIndex*: il suo valore viene aumentato solo se sono presenti esattamente 30 elementi, ovvero un secondo di registrazione è stato completato, i frame relativi al prossimo saranno inseriti nella posizione seguente. Coerentemente anche l'indice della lista dei frame viene azzerato, per effetto della funzione modulo, assicurando che dopo la fine di un secondo registrazione, il primo frame del successivo (sicuramente un frame di tipo I) venga inserito esattamente come primo elemento della lista.

La struttura dati riesce ad occupare una quantità di memoria pressochè costante e questo è dovuto principalmente alla dimensione fissa del numero di elementi che contiene. Infatti le uniche variazioni sono dovute dalla dimensione del relativo AVPacket che incapsula un puntatore ai dati grezzi forniti della video camera che sono dipendenti dalla scena che si sta riprendendo. La dimensione dei frame di tipo I può essere considerata costante poichè riferita ad un'immagine completamente descritta e non compressa, mentre quella dei frame P è variabile: in una scena in movimento i predictive frame conterranno molte differenze rispetto al frame completo dell'immagine e come risultato occuperanno più memoria. Invece analizzando la dimensione dei frame durante la registrazione di una scena ferma è possibile notare la poca memoria occupata dai predictive frame, che appunto contengono solo poche variazioni rispetto al frame precedente e quindi contengono un numero inferiore di informazioni al loro interno.

3.3.2 Creazione di un video

I dati contenuti nella struttura dati vengono utilizzati per la creazione del video in seguito ad una collisione, condizione scatenata dall'accelerometro che rileva una valore eccessivamente alto dell'accelerazione totale del veicolo. In questo scenario, è importante che il salvataggio dei dati di accelerazione e dei frame registrati avvenga contemporaneamente affinchè si riferiscano alla stessa finestra temporale e che possa combaciare l'istante del video in cui si vede l'urto con il valore più alto di accelerazione registrato dal sensore.

Questa sincronizzazione è particolarmente importante per assicurare la coerenza della prova che si vuole fornisce come dimostrazione del danno ricevuto: infatti senza i dati di accelerazione, non è possibile quantificare l'entità dell'urto e il risarcimento potrebbe non essere adeguato. L'unico modo per garantirlo è un meccanismo che permetta una comunicazione tra i due componenti software che si occupano della lettura dei dati del sensore e della ricezione dei frame dalla video camera, in modo da consentire l'inizio delle relative operazioni di salvataggio dei dati nello stesso momento: il costrutto signal/slot è particolarmente adatto in questa situazione. Durante la lettura dei dati dal sensore, in caso di urto rilevato, prima di procedere al recupero dei valori precedenti alla collisione viene emesso un signal e nel relativo slot viene avviata la stessa operazione sui frame, in modo che possano cominciare e terminare insieme.

Come nel caso dell'accelerometro, anche la ricezione dei frame dalla video camera viene gestita allo stesso modo, delegando ad un worker thread temporaneo le operazione di copia dei frame a seguito di un urto e utilizzando una condition variable per la sincronizzazione con il thread che continuamente riceve nuove immagini dalla video camera. Vale la pena far notare, che durante la normale ricezione dei frame, il thread si occupa semplicemente di posizionarli correttamente all'interno del buffer circolare nella posizione *currentIndex* ed aggiornarla di conseguenza, mentre durante la copia dei frame, a seguito di una collisione, ci sono delle operazioni aggiuntive da svolgere: in questo caso ci sono due thread che operano ed è quindi necessario sincronizzarli. La struttura diventa, infatti, una risorsa condivisa e bisogna garantire la validità dei dati sia in fase di lettura che di scrittura, per evitare che possano essere sovrascritti compromettendone il valore.

Salvataggio dei frame in seguito ad una collisione

In base a quanto descritto durante la copia dei dati di accelerazione, è evidente che in seguito ad una collisione, le strutture dati devono consentire la possibilità a due thread di accedere in mutua esclusione alle informazioni in esso contenute. Questa fase è infatti molto importante per quanto riguarda l'integrità dei dati che vengono mostrati all'utente: una sovrascrittura accidentale delle informazioni può portare all'invalidamento del video o delle accelerazioni memorizzate. Nella sezione 3.2.3 è stato presentato il procedimento di copia dei valori di accelerazione, mostrando come è stato sincronizzato l'accesso alternato alla struttura da parte dei due thread. In particolare, è stato analizzato solo il caso in cui il workerThread debba attendere affinchè l'altro thread legga i dati più recenti dall'accelerometro. Non è stato invece discusso il caso opposto, ovvero l'eventualità che il thread debba aspettare che il workerThread abbiamo terminato di copiare un certo dato dalla struttura dati.

Aspettare la copia dei dati

Il worker thread lavora anche in questo caso su un intervallo temporale che inizia prima e finisce dopo *currentIndex*, delimitato da *videoStartIndex* e *videoEndIndex*. Durante la copia, il thread che legge i dati deve evitare di interferire con l'intervallo del worker thread ed in particolare, currentIndex deve in ogni caso, riferirsi ad una posizione precedente rispetto a videoStartIndex. In caso contrario, i dati relativi agli istanti precedenti all'urto, verrebbero sovrascritti dai nuovi frame letti dalla video camera, compromettendo il video generato. Questa situazione potrebbe capitare nel caso in cui il worker thread sia molto lento nella copia dei dati, dando tempo all'altro thread di ricevere un numero così elevato di secondi di registrazione da dover sovrascrivere i più vecchi.

Volendo fornire un esempio, ipotizzando un buffer circolare di 100 posizioni e di conseguenza altrettanti secondi di registrazione memorizzati, la creazione di un video di 60 secondi e il valore di currentIndex pari a 50, abbiamo: videoStartIndex alla posizione 20 e videoEndIndex 80. In questo contesto, per fare in modo che la situazione precedente si verifichi, il worker thread dovrebbe impiegare oltre 70 secondi per copiare il primo secondo di registrazione, affinchè currentIndex raggiunga il suo stesso valore. Nonostante la bassa probabilità che possa accadere, se non in caso di mal funzionamento, si è comunque preferito aggiungere un meccanismo di prevenzione per assicurare il corretto funzionamento in ogni condizione.

La soluzione si basa sempre sul concetto di sincronizzazione tra thread, con l'utilizzo della condition variable per gestire l'attesa o la possibilità di continuare l'esecuzione dei thread. Da quanto detto, l'inserimento di un nuovo frame, discusso nella sezione precedente, viene effettuato in maniera differente durante la copia da parte del worker thread o meglio, viene preceduto da un controllo che stabilisce se la posizione in cui deve essere inserito è valida oppure deve ancora essere letta. Questa condizione viene verificata controllando il valore degli indici usati da entrambi i thread per effettuare le loro operazioni sul buffer ed in particolare si impone il vincolo che currentIndex preceda videoStartIndex.

In caso positivo, il thread che riceve dalla video camera è consapevole del fatto che quella locazione del buffer è stata già copiata dal worker thread e quindi può procedere sovrascrivendola con i dati più recenti. In caso contrario, i dati non sono stati ancora letti dal worker thread ed è necessario aspettare che concluda l'operazione, mettendo l'altro thread in uno stato di attesa. La sincronizzazione tra queste due entità è totalmente basata su currentIndex e videoStartIndex, i cui valori vengono modificati una un thread e letti dall'altro ed il loro cambiamento viene notificato tramite la condition variable che consente ad entrambi i thread di proseguire la loro esecuzione senza interferire con le posizioni della struttura in uso dall'altro. Durante la normale ricezione dei frame dalla video camera, invece, la condition variable non viene mai utilizzata dal thread perchè è l'unico ad accedere al buffer e può sovrascrivere le posizioni più vecchie senza alcuna conseguenza.

Il meccanismo appena descritto viene utilizzato anche sulla struttura dati che memorizza le accelerazione, ma si è preferito discuterne in questa sezione perchè è più probabile che accada durante la creazione del video, nonostante sia un evento molto raro. Infatti c'è una sostanziale differenza tra come i dati vengono recuperati:

- I valori di accelerazione vengono semplicemente copiati all'interno di un buffer temporaneo e questa operazione non è per nulla onerosa in termini di tempi di esecuzione.
- I frame della video camera, una volta copiati dalla struttura dati, vengono direttamente utilizzati per la creazione del video. Appena rilevata una collisione, FFmpeg genera un video mp4 vuoto che viene riempito poco alla volta dal workerThread. Questa operazione è sicuramente più costosa di una semplice copia ed inoltre non dipende solo dall'implementazione, ma anche da una libreria esterna. Per questo motivo si è ritenuto più sicuro considerare la possibilità di un rallentamento durante l'esecuzione e fornire un meccanismo per gestirlo.

3.4 Connettività

Fino ad ora sono state analizzate tutto le operazioni che il sistema effettua a seguito di un urto: creazione di un video e salvataggio dei valori di accelerazione. Questi dati sono salvati nella memoria locale del sistema che deve avere anche la possibilità di trasmetterli all'applicazione del proprietario, sfruttando le tecnologie di connettività supportate dal Raspberry. In questa fase ci si è soffermati sulle opzioni che si volevano garantire all'utente in modo da offrire la possibilità di poter sfruttare al meglio il sistema. Da questa analisi sono emerse due fondamentali funzionalità da realizzare:

- Notifica di una collisione.
- Possibilità di scaricare il relativo video e i dati di accelerazione.

Le notifiche sono una parte importante delle applicazioni odierne e vengono utilizzare per comunicare delle novità all'utente in maniera automatica e senza alcun intervento. Nel caso in analisi, il proprietario del veicolo non avrebbe alcun modo di notare la presenza di nuove informazioni all'interno del sistema, se non accorgendosi di un danno alla propria vettura. A questo punto può collegarsi con l'applicazione al Raspberry e scaricare i dati che riprendono la collisione. Ma ciò non si verifica se il danno è sufficientemente lieve da essere notato o non è visibile: in questo caso l'utente potrebbe accorgersene con molto ritardo. Per questo motivo si è deciso di sollevare il proprietario dall'onere di controllare manualmente la presenza di nuovi dati, fornendo al Raspberry la possibilità di comunicarlo all'applicazione.

La seconda funzionalità, invece, riguarda la fase in cui l'utente si è effettivamente collegato al sistema e vuole scaricare dei contenuti. In questo caso l'attenzione è stata posta sull'efficienza del trasferimento che deve essere immediato e terminare nel minor tempo possibile, per evitare lunghe attese. Le due funzionalità sono molto diverse tra di loro e richiedono un diverso ruolo da parte del Raspberry e dell'applicazione. Infatti nel caso di notifica, l'applicazione è in uno stato passivo, non richiede intervento da parte dell'utente e rimane in attesa aspettando di ricevere informazioni, mentre il dispositivo sul veicolo le invia in maniera attiva. Durante il download dei file, al contrario, è il Raspberry che aspetta richieste dall'applicazione ed in particolare dall'utente, che decide quali dati scaricare. Per l'implementazione

di queste funzionalità sono state utilizzate diverse tecnologie in base allo scopo: Bluetooth Low Energy(BLE) e Wi-Fi.

3.4.1 Bluetooth Low Energy

E'un protocollo di comunicazione basato sull'utilizzo di tecnologia wireless, che consente lo scambio di dati tra diversi device. A differenza del classico protocollo Bluetooth, il BLE è stato progettato per ridurre notevolmente i consumi energetici e questo lo rende adatto per essere utilizzati all'interno di sistemi embedded, inoltre è una tecnologia che si adatta perfettamente al sistema proposto in quanto il range di funzionamento del BLE è limitato ad alcuni metri. La comunicazione BLE è basata su un'architettura simile alla classica client-server, in cui i due dispositivi ricoprono uno specifico ruolo e vengono definiti central e peripheral. Tipicamente il peripheral fornisce i proprio dati agli altri dispositivi, il central utilizza i suoi dati per effettuare delle operazioni. Un classico esempio è rappresentato da un sensore di umidità che effettua delle misurazioni e le fornisce ad un'applicazione per smartphone.

Generalmente, il peripheral invia i dati in suo possesso in broadcast all'interno di pacchetti di advertising che possono essere visti come un sotto insieme dei dati che vuole inviare ad altri dispositivi, come informazioni relative al proprio hardware e ai servizi che è in grado di offrire (rilevamento della temperatura all'interno di una stanza). La comunicazione tra central e peripheral inizia proprio con l'advertising, che è l'unico modo che ha il peripheral per rendersi visibile: il central si limita ad effettuare scansioni o ascoltare tutti i pacchetti di advertising per trovare un dispositivo di interesse. Un dispositivo central può effettuare una richiesta di connessione verso un peripheral, con lo scopo di accedere alle informazioni in suo possesso. I dati del peripheral seguono questa struttura:

- Service: è l'insieme dei dati e delle funzioni che un peripheral può svolgere, ad esempio un rilevatore che fornisce la temperatura all'interno di una stanza utilizzando i dati provenienti dal sensore.
- Characteristic: fornisce informazioni aggiuntive rispetto al servizio esposto dal peripheral come la misura effettiva della temperatura o la posizione del sensore all'interno della stanza.
- Descriptor: informazioni accessorie sulle characteristics, come la possibilità di poter solo leggere o anche modificarne i valori.

Questo insieme di informazioni costituisce quello che nel mondo del BLE viene definito GATT Profile (Generic ATTribute Profile) che descrive il caso d'uso di un dispositivo, ad un esempio un Heart Rate Profile è indicativo dei dati e funzioni che il device è in grado di fornire. Normalmente su ogni peripheral è presente un GATT server che invia tramite advertising i servizi che espone, ognuno identificato univocamente da un UUID (Universally Unique IDentifier), una stringa standardizzata di 128 bits, formata da un valore di base combinato con uno specifico valore assegnato service predefiniti, il che facilita l'implementazione dei GATT Client (sul device central) che sono interessati a quel particolare service, che può così interrogare un certo servizio fornito dal GATT Server e leggerne le caratteristiche, anche esse associate a well-known UUID.

3.4.2 Advertising data

I pacchetti di advertising vengono inviati periodicamente da un device BLE e seguono una struttura ben definita. Lo stack Bluetooth riempie automaticamente molti dei campi presenti nel pacchetto, come l'indirizzo del mittente, il codice di controllo degli errori, il preambolo ed altre informazioni utili per l'instradamento e rappresentano l'header del pacchetto. Il payload è costituito da 32 byte, noti come advertising data e sono sotto il controllo dell'applicazione che può utilizzarli per i proprio scopi. Gli advertising data sono composti da una sequenza di strutture AD (Advertising Data), composte da un primo byte che specifica la lunghezza del dato (non considerando la propria dimensione), un secondo byte (AD Type) che specifica il tipo di dato che si vuole inviare e infine i dati effettivi (AD Data). Molti AD Type sono stati standardizzati ed elencati sul sito della Bluetooth SIG (Special Interest Group) [24]. Alcuni tra gli AD Type più utilizzati sono:

- 0x08 = Shortened Local Name con cui è possibile specificare il nome del dispositivo
- 0x0A = Tx Power Level per inviare il livello di batteria residua
- 0x01 = Flags che consente l'impostazione di parametri riguardo l'advertising, ad esempio è possibile impostare una discoverable mode limitata o indefinita

Custom Advertising Data

La libreria Qt mette a disposizione la classe QLowEnergyController che fornisce l'accesso ai dispositivi Bluetooth Low Energy. Il Raspberry è stato impostato per ricoprire il ruolo di peripheral nella comunicazione, in quanto dispositivo in possesso di dati da fornire allo smartphone e l'advertising è di tipo ADV_NONCONN_IND: specifica che non sono consentite connessioni verso il dispositivo e che l'advertising non è diretto ad uno specifico device ma è in modalità broadcast. La tecnologia Bluetooth è stata utilizzata esclusivamente per notificare all' applicazione Android la presenza di un nuovo video, rendendo superflua l'implementazione di un GATT Server sul Raspberry per fornire una semplice informazione. Infatti per lo scopo è stato sufficiente sfruttare la possibilità di inserire dati all'interno dei pacchetti di advertising che vengono mandati in broadcast e quindi facilmente accessibili da Android. Inoltre è stato possibile evitare la procedura di instaurazione della connessione tipica del Bluetooth, che comprende il pairing tra i dispositivi, riducendo la complessità dell'implementazione.

All'interno degli advertising data viene inserito il numero dei nuovi video disponibili sul Raspberry, che viene incrementato a seguito di ogni urto. L'evento appena descritto è stato realizzato con il sistema dei signal e slot forniti da Qt: una volta raccolti tutti i dati relativi al video, viene emesso un segnale, collegato ad uno slot che provvede a cambiare il valore del dato nel pacchetto di advertising. Con questo semplice sistema l'applicazione utente può facilmente ricevere i pacchetti Bluetooth, controllare il numero delle nuove collisioni ed eventualmente notificare all'utente la presenza di un nuovo video. Il problema a questo punto è il modo con cui l'applicazione può distinguere un pacchetto proveniente proprio dal Raspberry, piuttosto che da un altro dispositivo. Il meccanismo di advertising e ricezione dei pacchetti, non rappresenta una vera e proprio comunicazione tra le due parti che

non hanno alcun modo la possibilità di riconoscersi, senza ulteriori informazioni in loro possesso. Questo è proprio ciò che è stato realizzato per agevolare l'applicazione nell'identificazione del Raspberry: fornire dei dati univoci per distinguere i suoi pacchetti di advertising.

Identificare il Raspberry

Una tra le possibili soluzioni poteva prevedere l'inserimento di una sequenza fissa di valori all'interno del pacchetto, in maniera da essere facilmente riconoscibile. Questo però avrebbe costretto l'applicazione ad esaminare il contenuto di ogni pacchetto, il cui numero potrebbe essere molto elevato a causa della presenza di molti dispositivi low energy nelle vicinanze e della periodicità dell'operazione di advertising. Senza conoscere il mittente, sarebbe infatti necessario accedere al payload di ogni pacchetto, controllare la sequenza di valori scelta per identificare il Raspberry e leggere il numero dei video. Chiaramente questa soluzione, seppur funzionante, avrebbe richiesto un continuo lavoro da parte dell'applicazione, obbligata a controllare il payload ogni singolo pacchetto, prima di accorgersi che andava scartato, perchè non proveniente dal Raspberry. La computazione necessaria per questa operazione non è sicuramente rilevante, ma è senza dubbio un meccanismo che può essere reso molto più efficiente.

MAC Address Bluetooth

La soluzione scelta prevede, infatti, un sistema che consente all'applicazione di filtrare i pacchetti di advertising, in modo da poter facilmente individuare solo quelli di interesse. Questo è ciò che fa anche la sequenza fissa di valori, ma richiede l'analisi del pacchetto. Si è invece pensato ad un'informazione univoca del Raspberry che possa aiutare l'applicazione a riconoscerlo in maniera molto più immediata: l'indirizzo MAC dell'adapter Bluetooth che è unico per ogni dispositivo. Questo dato è infatti presente nell'header del pacchetto, impostato dallo stack Bluetooth ed è accessibile direttamente in fase di ricezione, evitando l'analisi del payload.

Android permette infatti di effettuare una scansione dei dispositivi Bluetooth, con la possibilità di impostare come filtro solo gli indirizzi a cui si è interessati, così che siano gli unici ad essere ricevuti. In questo modo l'applicazione effettua molto meno lavoro, dovendo leggere il payload solo dei pacchetti provenienti dal Raspberry. Come ulteriore ottimizzazione, la fase di advertising viene avviata e fermata sulla base del numero dei nuovi video da notificare, per evitare di comunicare l'assenza di nuovi dati, informazione inutile dal punto di vista dell'applicazione. Tutti questi accorgimenti hanno portato dei benefici in termini di efficienza ed immediatezza su entrambi i dispositivi, che si scambiano informazioni solo quando necessario:

- Il Raspberry si preoccupa di iniziare la fase di advertising solo se necessario, ovvero se ci sono nuove collisioni, modificando il valore all'interno del payload e dinamicamente fermarla dopo che l'utente ha avuto accesso all'insieme completo dei video memorizzati.

- L'applicazione riceve continuamente i messaggi di advertising e su questo non c'era la possibilità di intervenire, a meno di impostare degli intervalli

regolari in cui la ricezione viene interrotta e ristabilita, correndo però il rischio di ricevere i pacchetti con un certo ritardo. L'ottimizzazione riguarda in maniera specifica la ricezione, che si disinteressa completamente dei pacchetti provenienti da indirizzi MAC diversi da quello specificato, utilizzato come filtro. Questo permette all'applicazione di ricevere solo i pacchetti inviati dal Raspberry ed in particolare solo in presenza di nuovi video da notificare.

Il numero dei video all'interno del pacchetto viene incrementato in seguito ad ogni collisione, dopo la creazione sia del video che dei valori di accelerazione. Questo valore non è sempre crescente, infatti è prevista una condizione che ne provoca l'azzeramento ed è relativa ad uno scenario in cui l'utente è a conoscenza di tutti i video presenti sul Raspberry, rendendo superflua la notifica. Infatti, l'applicazione ha anche la possibilità di visualizzare la lista completa dei video memorizzati al suo interno che comprende anche quelli che avrebbe comunicato tramite advertising; a seguito di ciò il Raspberry interrompe l'invio dei pacchetti Bluetooth perchè l'utente ha appena avuto accesso a tutti video, compresi i più recenti.

La domanda a questo punto sorge spontanea, come viene comunicato all'applicazione il MAC address dell'adapter Bluetooth? Senza il pairing non è possibile instaurare una comunicazione Bluetooth, per cui questo canale di trasmissione non può essere utilizzato. Il Raspberry supporta però anche la comunicazione tramite Wi-Fi, che come descritto nella prossima sezione, viene usata per il trasferimento dei dati riguardanti le collisione. E' stato pensato di sfruttarla anche per inviare il MAC Address, per fare in modo che l'applicazione possa memorizzare questa informazione e filtrare i pacchetti di advertising. Il Wi-Fi ha quindi una duplice funzione:

- Consentire al Raspberry di inviare all'applicazione il video e i dati di accelerazione precedentemente creati.
- Ottimizzare la fase di advertising, specialmente dal punto di vista dell'applicazione Android che senza il MAC Address dovrebbe analizzare tutti i pacchetti, senza conoscere a priori quelli provenienti dal Raspberry.

Memorizzare il numero delle nuove collisioni

La presenza di nuovi video è da considerare una delle informazioni più importanti all'interno del sistema perchè è il modo con cui, tramite l'applicazione, viene informato il proprietario del veicolo dell'accaduto. Il numero dei video deve quindi essere preservato anche a seguito di uno spegnimento del sistema, per evitare che il suo valore venga resettato ad ogni avvio, il che porterebbe ad un comportamento errato del Raspberry che non comunicherebbe le giuste informazioni all'applicazione.

Il sistema deve essere in grado di aggiornare questo valore e memorizzarlo in fase di spegnimento, in modo da poterlo ripristinare al successivo avvio, per garantirne la coerenza. Senza questo meccanismo, in seguito ad ogni accensione il Raspberry comunicherebbe un valore nullo all'interno dei pacchetti, anzi, non lo trasmetterebbe affatto perchè lo considera come un'assenza di nuove informazioni; questo rappresenta chiaramente un comportamento errato e deve essere evitato. Per memorizzare questo dato, così come altri necessari al sistema, è stato utilizzato un file di testo serializzato in json, il cui contenuto viene recuperato ad ogni avvio

del sistema per ripristinare lo stato della precedente sessione di attività. Inoltre, il suo contenuto viene aggiornato non solo in fase di spegnimento, ma anche in seguito ad ogni cambiamento di stato del sistema, per avere la certezza che anche i dati più recenti vengano memorizzati.

3.4.3 Wifi

In questa sezione viene invece descritto come è stato realizzato il canale di trasmissione per l'invio dei dati dal Raspberry all'applicazione. La principale motivazione che ha portato all'utilizzo di una seconda tecnologia, oltre al Bluetooth, riguarda l'efficienza e il tempo di invio. Infatti, un'applicazione che utilizza il BLE ha un throughput di circa 1 Mbps e necessita di un certo quantitativo di tempo per il trasferimento dei video, che hanno una dimensione variabile ma dell'ordine di alcune decine di megabyte, influendo sulla user experience e sull'efficacia del sistema. Grazie alla presenza del modulo Wi-Fi integrato sulla scheda, è stato possibile configurare il Raspberry come Wireless Access Point. In questa modalità il dispositivo è in grado di funzione come un router e consente ad altri device di collegarsi in modalità wireless, autenticandosi con una password che verrà fornita all'utente, aggiungendo un ulteriore livello di protezione ai dati contenuti nel sistema.

3.4.4 Server

La comunicazione tra Raspberry e l'applicazione è stata realizzata secondo il classico paradigma client-server in cui il Raspberry ricopre ovviamente il ruolo di server, fornendo all'applicazione i dati che vengono richiesti. Per la realizzazione del server è stata estesa la classe QTcpServer per poter sfruttare tutte le funzionalità offerte da Qt. Un QTcpServer può operare in due diverse modalità, sincrona e asincrona. In modalità sincrona il server ascolta in maniera bloccante su un certo indirizzo ed una certa porta: ciò richiederebbe l'utilizzo di un thread separato per consentire agli altri componenti di continuare con la loro esecuzione. Invece, con la modalità asincrona il server può sfruttare il meccanismo dei signal e slot per essere notificata in caso di una nuova connessione da parte del client: questo però richiede l'esistenza di un event loop per rilevare il nuovo evento ed inviarlo allo slot registrato.

Inizialmente era possibile soddisfare una singola richiesta per volta, basandosi sul fatto che normalmente è solo l'applicazione del proprietario che interagisce con il Raspberry, ma in seguito si è deciso di gestire anche connessioni multiple, principalmente per consentire all'utente di scaricare più video contemporaneamente: ogni nuova richiesta viene quindi affidata ad un nuovo thread per l'esecuzione di trasferimenti paralleli. I dati inviati all'applicazione costituiscono una prova della collisione ricevuta e devono quindi essere trasferiti in maniera sicura per evitare che vengano intercettate o alterate da terzi, invalidandone ovviamente il valore. La soluzione proposta dal sistema è la realizzazione di un canale di trasmissione sicuro e cifrato fra le due parti per garantire l'autenticità e l'integrità delle informazioni durante l'invio all'applicazione.

TLS

Transport **L**ayer **S**ecurity è uno dei più diffusi protocolli crittografici ed è stato creato per garantire un livello di sicurezza alle comunicazioni tra peer su una rete internet. Usare un protocollo di trasporto sicuro assicura alcune proprietà aggiuntive al canale, che non possono essere garantire da un normale protocollo di trasporto, quale TCP/IP:

- Autenticazione tra le parti.
- Riservatezza dei messaggi.
- Autenticazione ed integrità dei messaggi.

L'autenticazione dei peer avviene tramite l'invio del certificato digitale X.509. All'interno della crittografia asimmetrica ogni partecipante alla comunicazione possiede una coppia di chiavi, pubblica e privata, con la seguente caratteristica: ciò che è stato cifrato con una delle due chiavi, può essere decifrato solo con l'altra. Il principale vantaggio della crittografia asimmetrica è proprio l'utilizzo di due chiavi con questa proprietà che assicura un livello di sicurezza maggiore.

Chiave pubblica e privata

La chiave privata deve essere conosciuta solo dal proprietario in quanto usate per la firma digitale, che garantisce al destinatario del messaggio autenticazione del mittente, non ripudio ed integrità: per questo il proprietario deve essere sicuro che la sua chiave non venga conosciuta da nessun altro. La chiava pubblica, invece, viene spesso distribuita perchè offre la possibilità di poter comunicare solo con il proprietario della relativa chiave privata. Infatti è spesso utilizzata per realizzare la riservatezza senza segreti condivisi, ovvero poter inviare un messaggio cifrato e segreto solo per uno certo destinatario che è anche l'unico che può decifrarlo, il detentore della privata appunto. Per questo la chiave pubblica viene indirettamente associata anche ad una persona o ad un'identità digitale in generale tramite il certificato X.509, un documento elettronico, firmato da una terza parte fidata detta CA (**C**ertification **A**uthority), che contiene dei dati relativi all'identità del proprietario e la sua chiave pubblica.

Il server può quindi autenticarsi inviando il proprio certificato e tramite la chiave pubblica ricevendo una sfida da parte del client: l'autenticazione del server è una fase obbligatoria del protocollo TLS, che, invece, può richiedere o meno il certificato del client per verificarne l'identità. Dopo la fase di autenticazione si deve provvedere a garantire la riservatezza dei messaggi: le informazioni trasmesse vengono generalmente cifrate utilizzando la stessa chiave, ovvero cifratura simmetrica, per una questione di efficienza dato che richiede meno computazione: le chiavi simmetriche sono infatti di dimensione decisamente inferiore rispetto alle chiavi asimmetriche e consente una cifratura più veloce dei messaggi.

Creazione della chiave simmetrica

La creazione della chiave simmetrica viene concordata tramite la cifratura asimmetrica, in modo tale che solo il possessore della rispettiva chiave privata possa

decifrarla e procedere alla cifratura dei messaggi. Infine l'integrità ed autenticazioni dei dati viene garantita da un keyed digest che consiste nell'applicazione ad ogni messaggio di una funzione di hash, basata sia sul messaggio stesso che su un segreto condiviso tra i peer. L'invio di un keyed digest fornisce al destinatario tutto le informazioni necessaria per assicurarsi l'integrità del messaggio, infatti in caso contrario gli hash sarebbero diversi ed inoltre la loro autenticazione: solo chi conosce il segreto può aver generato quel digest. Con l'instaurazione di un canale sicuro con TLS la comunicazione è considerata sicura tra le due parti, inoltre i messaggi sono cifrati per garantirne la riservatezza e qualsiasi eventuale manomissione durante il trasferimento può essere rilevata.

Certificato self-signed

Ritornando all'implementazione, Qt gestisce la creazione di canali sicuri dopo l'arrivo di una nuova connessione: la classe QTcpServer fornisce l'accesso al socket descriptor che identifica la connessione accettata ed è in questo punto che viene generato un nuovo thread che si occupa della trasmissione dei dati. Scendendo più nello specifico, il thread utilizza il socket descriptor per creare un oggetto di tipo QSslSocket, una classe di Qt che fornisce un socket cifrato e sicuro per la trasmissione dei dati. Per inizializzare il socket sono ovviamente necessari dei documenti che ne attestino l'identità, come la coppia di chiavi ed un certificato valido, per il quale è obbligatorio avere anche la firma digitale di una CA. Si è quindi optato per la creazione di un cosiddetto certificato self-signed, ovvero firmato dalla stessa identità che lo ha creato. Ovviamente non ha alcun tipo di valore, perchè nessuna terza parte può garantire per l'identità di questo peer, ad esempio nessun browser instaurerebbe una connessione cifrata con questo server proprio a causa della mancanza della firma di una CA.

Nonostante ciò, è comunque possibile instaurare una comunicazione sicura con l'applicazione del proprietario del veicolo perchè analizzando il server, è conforme con il protocollo TLS: possiede una coppia di chiavi ed un certificato. E' stato invece necessario "forzare" il client, rappresentato dall'applicazione, a fidarsi di un'entità e quindi di un certificato, senza alcuna CA come garante. Per lo stesso motivo, la richiesta di autenticazione del client è stata lasciata come opzionale, altrimenti sarebbe stato necessario un ulteriore certificato valido, oppure generarne un altro self-signed e "forzare" Qt a ricevere e gestire connessioni con un peer untrusted. Dopo l'inizializzazione del socket, comincia la fase di handshake, in cui le due parti si scambiati i parametri di sicurezza e la chiave simmetrica che verrà utilizzata per cifrare i dati all'interno del canale sicuro. Grazie ad un signal delle QSslSocket, è possibile conoscere il momento in cui l'handshake è finito: da questo istante in poi tutti i messaggi saranno cifrati e si può iniziare effettivamente il trasferimento dei dati.

Protocollo testuale

La comunicazione tra applicazione e Raspberry è di tipo request-response: l'applicazione invia una richiesta che corrisponde ad una risorsa che viene inviata. I

messaggi di richiesta sono dei semplice messaggi testuali che vengono letti e differenziano il tipo di dato richiesto dall'applicazione. Il Raspberry mette a disposizione le seguenti alternative:

- Presentazione: è il modo con cui il Raspberry si presenta all'applicazione, ovvero invia proprio MAC address Bluetooth, che verrà utilizzato per filtrare i pacchetti di advertising. Questa operazione si è rilevata necessaria in seguito alla decisione di usare diverse tecnologie per la comunicazione tra i due principali componenti del sistema. Le informazioni utili per il collegamento Wi-Fi sono considerate note a priori, in quanto fornite all'utente in fase di installazione del sistema, ma non sono sufficienti per il funzionamento delle notifiche in quanto l'applicazione non sarebbe in grado di distinguere il dispositivo. L'applicazione dovrebbe richiedere questa informazione durante la prima configurazione per assicurare una corretta comunicazione tra le parti ed è sufficiente che sia effettuata una sola volta.

- Lista video: con questa richiesta l'applicazione invia la lista completa dei video relativi ad urti ricevuti salvati nel sistema. Si è deciso di fornire questa informazione all'utente in modo da avere una visione completa delle informazioni presenti e per poter selezionare liberamente quale video scaricare. Dopo aver inviato la lista completa, viene anche azzerato il numero dei video da trasmettere nei pacchetti di advertising perchè l'utente può così notare la presenza di eventuali nuovi video, non ancora scaricati.

- Specifico video: l'applicazione può richiedere i dati di uno specifico video, che comprendono il video stesso ed altre informazioni aggiuntive, quali timestamp, durata del video e i dati di accelerazione relativi.

Anche in questo caso i dati vengono prima serializzati in json e poi inviati, in modo che l'applicazione possa facilmente deserializzati e trasformare la stringa ricevuta in uno specifico oggetto, in base alla risorsa richiesta.

3.5 Unire i dati

Finora sono stati analizzati i diversi moduli software che compongono il sistema, mettendo in risalto non solo il loro utilizzo, ma anche i dettagli di come stato stati realizzati. Ognuno di loro ha un compito ben preciso ed indipendente dagli altri, come la ricezione dei frame dalla video camera o l'invio dei pacchetti di advertising. Ciò che invece li accomuna è lo scopo finale: combinare i dati raccolti e fornirli all'utente in modo che possa rivedere ed analizzare la dinamica dell'accaduto. Ci sono, inoltre, ulteriori informazioni della collisione che devono essere aggiunti, come la data e l'orario ed opzionalmente la posizione del veicolo in quel momento. Quello che manca è il criterio con cui vengono associati questi dati, un'informazione che possa determinare che i valori di accelerazione corrispondano effettivamente ad un certo video in modo da poterli unire prima di renderli disponibili all'utente. Un'altra questione che non è stata ancora affrontata, riguarda la responsabilità di tutti i dati generati in seguito ad una collisione: i vettore delle accelerazioni, il video, il timestamp, etc. Quello che stiamo descrivendo è un componente software

che occupi una posizione centrale all'interno del sistema, con cui gli altri si possano interfacciare per il salvataggio o recupero dei dati.

Video Manager

Questo ruolo è ricoperto dal Video Manager che si occupa della gestione dei video, coordinandosi con gli altri componenti del sistema, precedentemente descritti. In questa sezione viene descritto l'interno insieme delle funzionalità svolte da questo componente, che lo rendono una parte fondamentale del sistema. Il Video Manager mantiene al suo interno tutte le informazioni riguardanti i video memorizzati e facilita gli altri componenti nella generazione e unione dei dati in seguito ad una collisione. Infatti, subito dopo un urto, crea le prime informazioni che caratterizzeranno il video: il nome ed il timestamp. Proprio il nome viene utilizzato come criterio per l'associazione dei dati appartenenti allo stesso video ed il Video Manager assicura che sia univoco all'interno del sistema, per evitare duplicati. Infatti, dopo che i rispettivi componenti software hanno completato la creazione del video e del vettore delle accelerazioni, il gestore mette insieme questi dati, tramite appunto il nome appena creato.

Ogni video è infatti modellato come un oggetto contenente le seguenti informazioni:

- Nome: utilizzato per identificare univocamente e per ricavare il percorso del file multimediale all'interno della memoria del Raspberry.
- Timestamp: tiene traccia del momento esatto in cui si è verificata la collisione.
- Durata: contiene la durata in secondi del video.
- Dati di accelerazione: sequenza completa dei valori rilevati dal sensore, prima e dopo l'urto.

Il gestore dei video mantiene in memoria, in ogni momento, la lista completa dei video precedentemente creati. Poichè il nome viene creato per essere unico, è stata utilizzata una mappa come struttura dati, in cui le informazioni dei video sono indicizzate in base al nome. Come descritto nella sezione x, il thread che si occupa della lettura dall'accelerometro, ha anche la responsabilità di segnalare un valore troppo alto, che identifica una collisione. In quello stesso momento, viene invocato il video manager che inizia la creazione di un nuovo oggetto da inserire nella mappa, con le informazioni presenti in quel momento: il nome del video che ha appena creato e il corrente timestamp. Da questo punto in poi, la struttura dati si troverò in una stato incompleto, una alcuni dati presenti e altri mancanti, perchè devono essere ancora generati dalla video camera e dall'accelerometro. Poichè vengono ripresi anche gli istanti successivi all'urto, ci vorrà del tempo affinchè sia pronti. Il video sarà completo, all'interno della mappa, quando entrambi i componenti avranno inserito i loro output al suo interno. Il problema a questo punto è trovare il momento esatto in cui sono stati generati sia il video che i valori di accelerazioni: solo quando sono disponibili queste informazioni l'elemento all'interno della mappa si può considerare completo e pronto per essere scaricato dall'applicazione.

In aggiunta, la nuova collisione deve anche essere notificata allo smartphone e questo avviene con l'utilizzo dei pacchetti di advertising. Il video manager si

occupa anche di questo compito: aggiornare il valore all'interno dei pacchetti in modo che l'applicazione possa riceverli ed accorgersi della presenza di nuovi dati. Questa responsabilità è stata affidata al video manager proprio per il suo ruolo centrale che ricopre. Infatti, è l'unico componente che gestisce i dati memorizzati sul Raspberry ed i nuovi che vengono prodotti dopo una collisione. E' evidente che il video manager abbia una conoscenza completa del sistema, non solo dal punto di vista dei dati, ma anche dalle connessioni che ha con gli altri componenti con cui comunica per svolgere le funzioni richieste dall'applicazione.

Notificare una nuova collisione

La notifica di una nuova collisione avviene dopo la ricezione del video e dei valori di accelerazione, perchè i precedenti dati sono stati già creati dal video manager nel momento in cui l'accelerometro ha rilevato un valore eccessivamente alto. L'aggiornamento dei pacchetti di advertising viene effettuato da un bluetooth manager che si occupa solo della gestione dell'adapter Bluetooth. I collegamento tra questi due componenti è stato realizzato nuovamente con il meccanismo dei signal e slot ed in particolare, il signal viene emesso solo dopo il salvataggio del video e dei dati di accelerazioni. La problematica principale di questa condizione è identificare il momento preciso in cui avviene, perchè ogni informazione viene fornita una thread diverso. Nello specifico, il thread che crea il video inserisce la durata del file, mentre il thread che copia i dati di accelerazione li trasmette al video manager, in modo che possa memorizzarli nella mappa. Il tempo che impiegano i thread per effettuare queste operazioni è molto simile, perchè devono recuperare i dati relativo allo stesso intervallo di tempo, ma non è detto che terminino contemporaneamente.

L'approccio utilizzato per la soluzione a questo problema è il seguente: ogni thread, dopo aver terminato le proprie operazioni di copia, inserisce i dati all'interno della mappa e controlla se i dati dell'altro thread sono presenti. In caso negativo, è il primo thread che ha terminato, quindi è a conoscenza del fatto che mancano ancora dei dati. In caso positivo, il thread sa di essere l'ultimo ad accedere alla mappa e che mancano solo le sue informazioni. Il principio alla base di questo meccanismo è fare in modo che l'ultimo ad inserire le informazioni si occupi anche di comunicare al bluetooth manager di aggiornare il valore nei pacchetti di advertising, perchè tutti i dati relativi al nuovo video sono stati memorizzati.

In questo scenario, la mappa diventa una risorsa condivisa, perchè entrambi i thread devono accedere ad essa per controllare la presenza delle informazioni e devono farlo in mutua esclusione, acquisendo un mutex. Senza un meccanismo di sincronizzazione, può capitare che entrambi credano di essere gli ultimi ad aver terminato perchè notano la presenza dei dati altrui. Questo deve essere assolutamente evitato perchè nessuno emetterebbe il signal per aggiornare il valore nei pacchetti di advertising ed l'applicazione non potrebbe conoscere l'esistenza di questa nuova collisione.

Anche il server interagisce con il Video Manager per recuperare le informazioni dei video che devono essere trasmessi all'applicazione:

- Per inviare la lista dei video, il server si fa restituire dal Video Manager la stringa json contenente le informazioni principali di ogni video memorizzato all'interno della struttura.
- L'invio del singolo video avviene in maniera simile, ma in più viene inviato il file multimediale e i dati di accelerazione.

Persistenza

Il gestore dei video, essendo l'unico componente in possesso della lista dei video, si occupa anche di salvarla su disco e questo è necessario per fare in modo che i nuovi dati generati dal Raspberry vengano preservati anche a seguito di un suo spegnimento, che accade generalmente ad ogni accensione del veicolo. I video generati vengono ovviamente scritti direttamente sul disco, ma ciò non è vero per le altre informazioni: la data di creazione e i dati di accelerazione verrebbero persi ad ogni avvio. Inoltre il Raspberry deve comunque essere a conoscenza della directory in cui vengono salvati ogni volta i video, per poterli recuperare e trasmettere all'applicazione. A questo proposito è stato utilizzato per file di testo contenente tutte le informazioni riguardo i video completi memorizzati nel sistema e viene modificato nelle seguenti occasioni:

- Ad ogni avvio il Raspberry legge l'intero contenuto del file, la cui locazione è nota, recupera le informazioni dei video e le carica all'interno della mappa. La struttura dati viene mantenuta in memoria RAM per velocizzarne l'accesso durante l'esecuzione, specialmente nella fase di trasmissione all'applicazione. Infatti nel caso in cui l'utente dovesse richiedere la lista dei file, il Raspberry dovrebbe prima recuperarla da file di testo, rallentando l'operazione.
- Dopo ogni urto e la relative generazione dei dati il file viene aggiornato di conseguenza, aggiungendo le nuove informazioni disponibili.

Oltre ai dati sui video, all'interno del file json, viene memorizzato anche il numero delle collisioni da annunciare all'applicazione, così che il suo valore venga preservato anche in seguito allo spegnimento del Raspberry.

Da quanto appena descritto, è evidente il ruolo fondamentale del gestore che ricopre una posizione centrale nella logica del sistema, specialmente nella fase compresa tra la generazione di nuovi dati ed il momento in cui vengono inviati all'utente ed inoltre fornisce l'accesso ai video memorizzati che rappresentano la risorsa fondamentale del sistema intero. Tutto ciò si trasforma nella necessità da parte degli altri componenti di poter comunicare con il gestore ed ottenere medesime informazioni, ad esempio la stessa lista dei video o lo stesso nome del video che viene creato dopo la collisione. Da un punto di vista implementativo, è possibile garantire tutto questo con l'utilizzo di un Singleton per la realizzazione del gestore dei video.

Singleton

Il Singleton è un design pattern creazionale specifico della programmazione ad oggetti. Con il suo utilizzo si vuole garantire la creazione di una singola istanza per una classe e un punto di accesso globale a questa istanza. Il Singleton viene spesso usato in presenza di risorse condivise tra più classi, come la connessione ad

un database o la gestione di un file di log, che possono accedervi tramite l'unica istanza disponibile. E' quindi compito della classe stessa gestire la propria istanza, assicurarsi che sia l'unica ed allo stesso tempo fornirne l'accesso globalmente. Tipicamente questo viene realizzato dichiarando tutti i costruttori del Singleton privati, in modo che non ne possano essere create altre istanze al di fuori della classe e dichiarando un metodo statico pubblico che restituisca un riferimento all'istanza.

3.6 Implementazione Android

L'altro componente fondamentale del sistema è l'applicazione Android, fornita all'utente per poter usufruire di tutte le funzionalità messe a disposizione dal Raspberry ed è l'unico punto di accesso alle informazioni memorizzate al suo interno. La realizzazione dell'applicazione Android è stata pensata per essere semplice in modo da non limitarne l'utilizzo solo ad utenti con esperienza, ma allo stesso tempo efficiente per poter sfruttare tutte le potenzialità del sistema. L'interfaccia è infatti immediata e priva di elementi grafici superflui per evitare confusione sulle operazioni da effettuare.

3.7 Interfaccia generale

L'interfaccia generale dell'applicazione è stata pensata per fornire all'utente una visione chiara e semplice di tutti i dati presenti all'interno del sistema. Inoltre offre la possibilità di poter scaricare i video e i relativi dati sullo smartphone, per averne il pieno controllo. Queste motivazioni hanno portato alla creazione della principale schermata dell'applicazione, formata sia dalla lista dei video precedentemente scaricati e presenti nello smartphone, che dalla lista dei video memorizzati sul Raspberry disponibili per il download. Di seguito viene illustrato il funzionamento delle due schermate appena descritte e delle possibili interazioni con l'utente e con il sistema installato sul veicolo. L'activity principale dell'applicazione contiene al suo interno un View Pager, un componente grafico che consente all'utente di navigare tra diverse pagine di dati ed è spesso utilizzato insieme ai fragment, perchè semplifica la gestione del loro ciclo di vita. I fragment sono stati utilizzati per mostrare all'utente sia la lista dei video precedentemente scaricati che quelli presenti sul Raspberry.

Sulla base delle funzionalità che l'applicazione offre all'utente, sono state realizzate 4 schermate di interesse:

- Visualizzazione di tutte le reti Wi-Fi presenti in zona, tra cui l'utente può facilmente individuare quella creata dal Raspberry e connettersi ad essa.
- Lista dei video precedentemente scaricati e memorizzati all'interno dello smartphone.
- Lista dei video presenti sul Raspberry che possono essere selezionati e scaricati dall'applicazione.
- Visualizzazione effettiva del video e del grafico che mostra i valori di accelerazione durante la collisione.

3.7.1 Service

L'utilizzo del Bluetooth Low Energy da parte del Raspberry ha richiesto l'implementazione della stessa tecnologia all'interno dell'applicazione: questo è l'unico vincolo imposto dall'applicazione per il suo corretto funzionamento, infatti è una funzionalità disponibile sui dispositivi Android a partire dalla versione 4.3 e ne esclude l'utilizzo su smartphone con versioni precedenti del sistema operativo.

Le API a supporto del Bluetooth Low Energy consentono allo smartphone di scoprire e comunicare con altri dispositivi BLE che hanno limitazioni sul consumo energetico, come sensori e sistemi embedded in generale. Lo smartphone ricopre, invece, il ruolo di central all'interno della comunicazione Bluetooth, ed è quindi il dispositivo che riceve dati dal peripheral per analizzarli. La notifica di un nuovo video all'interno di un pacchetto di advertising, rappresenta un evento non deterministico e per questo è necessario ascoltare continuamente i pacchetti in arrivo dal Raspberry. Per questo tipo di operazione è stato utilizzato un Service, componente applicativo di Android che consente l'esecuzione di lunghi task in background e non fornisce interfaccia grafica. Il grande vantaggio dell'utilizzo di un service è la possibilità di effettuare operazioni anche quando l'applicazione non è in foreground o non è in esecuzione che assicura una maggiore usabilità ed efficienza. Android consente la creazione di tre tipi di service:

- Foreground: è un service con l'utente può interagire e per questo deve fornire una notifica all'interno della status bar. E' solitamente utilizzato per la riproduzione di musica video o per il download di file.
- Background: un background service può essere eseguito senza l'intervento dell'utente, come la ricezione di email o chiamate.
- Bound: gli altri componenti di Android possono effettuare un'operazione di binding con il service e instaurare una comunicazione di tipo client-server con cui richiedere o ricevere informazioni. Quando non c'è più nessun collegamento con gli altri componenti, il service viene distrutto.

A differenza dei service bound, il cui ciclo di vita dipende da quello dei componenti a cui è legato, gli altri due tipi possono essere eseguiti per un tempo indefinito, fino a che non vengono esplicitamente fermati dall'applicazione. Oppure la loro terminazione può essere forzata da Android in caso di mancanza di memoria, in modo da recuperare risorse da favorire l'applicazione in foreground. I background service sono i primi ad essere candidati per la terminazione forzata in quanto non forniscono alcun tipo di interazione con l'utente che spesso non è nemmeno a conoscenza della loro esecuzione. Dopo aver recuperato le risorse necessarie, Android prova a far ripartire il service, precedentemente chiuso, se richiesto esplicitamente, così da consentirne l'esecuzione indipendentemente dallo stato dall'applicazione.
Il service realizzato all'interno dell'applicazione ha come unico scopo quello di ascoltare i pacchetti di advertising provenienti dal Raspberry ed eventualmente notificare l'utente della presenza di un nuovo video.

Configurazione del Bluetooth Low Energy

Per accedere all'adapter Bluetooth del dispositivo, Android mette a disposizione l'oggetto *BluetoothAdpter* che permette di svolgere le comuni operazioni tipiche del

Bluetooth, come la ricerca di dispositivi, la possibilità di effettuare il pairing e la creazione di specifiche socket per l'invio e ricezione di dati. Per la gestione del Bluetooth Low Energy è invece necessario utilizzare la classe *BluetoothLeScanner* che consente all'applicazione di scoprire nuovi dispositivi. Per iniziare una nuova scansione è possibile impostare dei filtri sulle principali caratteristiche del dispositivo di interesse, quali nome, MAC address, gli UUID associati ai servizi esposti dal server GATT.

Inoltre si possono associare degli specifici parametri per il genere di scansione che si vuole effettuare, tra cui la frequenza di ricerca che può essere impostata per garantire un basso consumo energetico per limitare l'utilizzo della batteria dello smartphone. Da Android 8.1, infatti, le scansioni senza filtri vengono automaticamente bloccate quando lo schermo del dispositivo si spegne, un'ottimizzazione stabilita dal sistema operativo per aumentare il tempo di utilizzo.E' inoltre obbligatorio che l'applicazione abbia i permessi per accedere alla posizione dello smartphone e che il GPS sia abilitato per effettuare correttamente una scansione, un altro vincolo imposto dalla versione 6.0 di Android.

I risultati della scansione vengono comunicati all'applicazione tramite delle callback, in particolare la funzione *onScanResult* viene chiamata a seguito di ogni advertisement e ha come parametro uno *ScanResult* che contiene tutte le informazioni relative al pacchetto ricevuto. In particolare, l'applicazione è interessata esclusivamente a leggere i dati presenti all'interno del payload del pacchetto, che contengono il numero dei nuovi video registrati dal Raspberry ed eventualmente avvisare l'utente con una notifica.

Gestione delle notifiche

A questo punto è stato necessario implementare un meccanismo per evitare l'invio di notifiche multiple: infatti il service gira esclusivamente in background sullo smartphone e quando rileva un numero maggiore di zero di video, sa che deve notificare l'utente. Il problema che emerge da questa situazione è evidente considerando la periodicità dell'advertising e la scansione dell'applicazione che deve essere sempre attiva per evitare di perdere i pacchetti. Ipotizzando la presenza di un nuovo video, il Raspberry inizia la fase di advertising specificando il valore 1 all'interno del payload del pacchetto: lo smartphone, in prossimità del Raspberry, riceve il pacchetto, nota un valore maggiore di zero e crea una notifica per l'utente, comunicandogli la presenza di nuove informazioni. Dopo qualche secondo, il Raspberry invia nuovamente lo stesso pacchetto che porta alla ripetizione delle operazioni appena descritte.

La radice di questo problema va ricercata nelle scelte implementative effettuate per la comunicazione che hanno portato ad utilizzare la tecnologia Bluetooth solo per l'advertising ed il Wi-Fi per l'invio dei video.

- Sfruttare il pairing Bluetooth per creare una connessione di tipo client-server basata su socket, avrebbe consentito all'applicazione di comunicare al Raspberry la ricezione del messaggio di advertising, evitando l'invio dello stesso valore all'interno del successivo pacchetto. Il principale vantaggio di questa soluzione è l'utilizzo di una singola tecnologia per la creazione di un canale di comunicazione e quindi alla riduzione della complessità del sistema in generale. Inoltre, il pairing tra dispositivi Bluetooth è un'operazione spesso più

intuitiva e rapida rispetto al collegamento ad una rete wireless, ma questa soluzione è stata scartata a causa della limitata velocità della trasmissione con il Bluetooth Low Energy, che avrebbe richiesto dei tempi molto lunghi per l'invio dei video, obbligando l'utente a rimanere in prossimità del sistema fino alla conclusione dell'operazione.

- Una connessione Wi-Fi, invece, riduce drasticamente le tempistiche per la trasmissione di file, ma non sarebbe stata adatta per accorgersi della presenza di un nuovo video, infatti lo smartphone dovrebbe essere sempre connesso alla rete del Raspberry per poter ricevere i dati. Vale la pena far notare che il Raspberry non è in grado di fornire l'accesso ad internet, per cui instaurare connessioni periodiche all'access point, anche solo per verificare la presenza di nuovi dati, limiterebbe l'utilizzo della connessione dati da parte dell'utente: per queste motivazioni è stato preferito un utilizzo ibrido della due tecnologie.

Ritornando alla precedente problematica, è stata adottata una soluzione basata sulla memorizzazione del numero di video da notificare all'utente. Lo scenario di funzionamento cambia leggermente: il Raspberry inizia una fase di advertising specificando il valore 1 all'interno del pacchetto, il service notifica l'utente e memorizza il numero appena ricevuto. Nelle successive interazioni, prima di inviare la notifica, l'applicazione controlla se il valore all'interno pacchetto è superiore a quello memorizzato:

- In caso negativo il service non effettua alcuna operazione. Questa condizione indica che l'utente è stato già avvisato della presenza di un nuovo video ed ulteriori notifiche sarebbero ridondanti.

- In caso positivo, invece, il Raspberry ha rilevato una nuova collisione ed è necessario notificarlo all'utente. L'applicazione procede poi all'aggiornamento del numero dei video notificati.

Con questo meccanismo il service è in grado di distinguere le informazioni nuove da parte del Raspberry da quelle già ricevute, evitando di inviare notifiche superflue. L'azzeramento del valore memorizzato e del numero di video da inserire nei pacchetti, viene effettuato dopo la ricezione, da parte dell'applicazione, della lista completa dei video presenti sul Raspberry. L'utente infatti può accorgersi personalmente della presenza di nuovi collisioni, senza il bisogno di essere notificato.

3.7.2 Connessione con il Raspberry

Il sistema, nel suo complesso, comprende anche l'applicazione realizzata, che è un supporto per l'utente che può accedere ai dati presenti sul Raspberry ed è quindi necessario instaurare una comunicazione tra le due parti. Come precedentemente accennato, il Raspberry funziona anche da access point ed espone un rete Wi-Fi a cui è possibile collegarsi utilizzando un SSID e una password, forniti all'utente nel momento dell'installazione del sistema. Questa operazione è infatti identica al collegamento di uno smartphone al router di casa, di cui si conoscono le credenziali. Per questo motivo si è deciso di seguire le implementazioni dei più comuni sistemi operativi per la gestione della comunicazione wireless. Infatti, l'applicazione mostra la lista delle reti Wi-Fi nelle vicinanze tra cui l'utente può selezionare quella

relativa al sistema e inserire la corrispondente password, come visibile in figura 3.2: Android tiene traccia della connessione appena effettuata e memorizza le credenziali di accesso in modo da non doverle richiedere in futuro.

Figura 3.2. Reti wifi nelle vicinanze

Per ottenere la lista dei video è stato implementato un Broadcast Receiver, utilizzato per ricevere i risultati della scansione effettuata dalla scheda di rete dello smartphone.

Broadcast Receiver

Un Broadcast Receiver è un componente di Android che consente ad un'applicazione di essere notificata in seguito ad un particolare evento. A differenza degli altri componenti del sistema operativo, un Broadcast Receiver è privo di interfaccia grafico, ma ha la possibilità di generare notifiche nella status bar di Android. Il sistema operativo invia autonomamente numerosi messaggi in broadcast, che possono essere intercettati dalle applicazioni, in modo che possano reagire nella maniera più opportuna. Alcuni esempio sono: batteria scarica, chiamata o sms in arrivo, stato della connessione, etc. Il Broadcast Receiver comunica con l'applicazione tramite la callback

```
onReceive(Context context, Intent intent)
```

che viene invocata ogni volta che si verifica l'evento registrato dal receiver. La funzione fornisce come parametro un oggetto di tipo Intent, da cui è possibile ricavare

numerose informazioni riguardo l'evento appena catturato. Un Broadcast Receiver deve essere "registrato" per poter essere notificato dal sistema operativo, che altrimenti non saprebbe niente della sua esistenza. Con un normale registrazione, il receiver riporterà all'applicazione ogni genere di evento che gli viene comunicato da Android, anche quelli a cui non si è interessati. Il tipo di evento può sempre essere determinato dal relativo intent, così che l'applicazione possa distinguerli e reagire di conseguenza: questo funzionamento è però poco efficace, perchè costringe il receiver ad ascoltare tutti gli eventi ed esiste infatti, un meccanismo migliore per filtrarli. Un'applicazione può dichiarare il suo interesse solo a specifici eventi con un IntentFilter, che verrà utilizzato dal receiver per filtrare gli eventi ricevuti. L'IntentFilter deve essere inizializzato una stringa, che rappresenta un'azione e quindi l'evento che si vuole intercettare.

Per ottenere la lista delle reti, è stato registrato un evento di tipo: WifiManager.SCAN_RESULTS_AVAILABLE_ACTION che notifica l'applicazione ogni volta che è terminata una scansione delle reti disponibili. Ogni volta che si verifica, è possibile interrogare il WifiManager, un oggetto che gestisce l'adapter Wi-Fi dello smaprthone, per ricevere la lista delle reti trovate nelle vicinanze e mostrarle all'utente.

3.7.3 Configurazione del Raspberry

Il collegamento alla rete wifi è solo la prima parte della configurazione, infatti il suo scopo è far conoscere una nuova rete al sistema operativo, ma ciò che manca è quella che è stata definita una "presentazione", ovvero una procedura di configurazione che consente all'applicazione di riconoscere il Raspberry del proprietario del veicolo. Come discusso in precedenza, il MAC address Bluetooth del Raspberry viene utilizzato per distinguere i pacchetti di advertising in modo da ottimizzare le scansioni dall'applicazione e poter accedere solo a quelli di interesse e l'invio di questa informazione avviene proprio a seguito della connessione alla rete wifi.

L'ipotesi alla base di questa operazione è la conoscenza da parte dell'utente delle credenziali per accedere alla rete, che riconosce il proprio Raspberry tra la lista delle reti e si collega. Come ulteriore garanzia, per essere sicuri di star comunicando con il giusto dispositivo, l'applicazione invia una semplice richiesta di presentazione testuale e si aspetta una specifica risposta. Il Raspberry invia due fondamentali informazioni: il MAC address dell'adapter Bluetooth e l'identificativo SSID della propria rete Wi-Fi.

Solo a questo punto, l'applicazione avvia il service che può iniziare a filtrare i pacchetti di advertising: senza il MAC address non sarebbe possibile riconoscere il Raspberry tra tutti gli altri dispositivi BLE ed il service non sarebbe di alcuna utilità. Come accennato precedentemente, dalla versione 8.1 di Android, le scansioni non filtrate vengono automaticamente bloccate quando lo schermo dello smartphone si spegne, impedendo al service di ricevere nuove informazioni dal Raspberry. Per evitare che ciò accada, tutte le scansioni utilizzano il MAC address come filtro e questo permette all'applicazione di:

- Evitare il controllo sul MAC address del pacchetto ricevuto per verificarne la provenienza perchè la scansione viene filtrata direttamente dall'adapter dello smartphone.

- Notificare la presenza di nuovi video anche quando l'utente non utilizza lo smartphone, limitando al minimo la possibilità di perdere i pacchetti inviati dal Raspberry.

L'identificativo della rete viene utilizzato dall'applicazione riconoscere l'access point creato dal Raspberry e per controllare lo stato della connessione. Android associa infatti ad ogni rete wifi configurata un id, che è possibile recuperare fornendo il relativo SSID e viene utilizzato per collegarsi alle rete: questo consente all'applicazione di collegarsi dinamicamente alla rete del Raspberry senza l'intervento dell'utente, automatizzando il processo di ricezione dei dati. La connessione con il dispositivo avviene solo in seguito a specifiche richieste da parte dell'utente e come tale, è un'operazione che ha una durata limitata. Per questo motivo, dopo uno scambio di dati, la connessione viene forzatamente interrotta, ricollegando lo smartphone alla precedente rete wifi: questo perchè il Raspberry non fornisce accesso ad internet e l'utente dovrebbe manualmente cambiare la sua connessione. In queste situazione è spesso Android che interviene, notificandolo all'utente, ma per una migliore usabilità è stato preferito implementare il cambio automatico di rete, nel caso in cui il sistema operativo non dovesse provvedere.

3.7.4 Lista dei video sullo smartphone

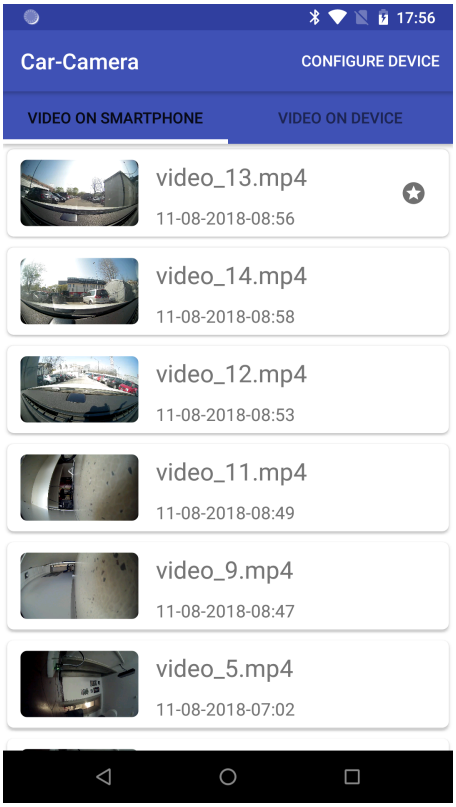

Figura 3.3. Lista dei video memorizzati sullo smartphone

La figura 3.3 mostra tutti i video scaricati dal Raspberry che vengono salvati nella memoria interna dello smartphone per essere sempre a disposizione dell'utente. La prima decisione che è stata presa in merito ai video sullo smartphone riguarda la posizione in cui memorizzarli. L'utilizzo della directory di default usata da Android per i file multimediali sarebbe stata sicuramente una valida opzione, ma avrebbe mischiato i dati personali dell'utente con i video generati dal sistema e sicuramente complicato la loro ricerca. Si è quindi preferito utilizzare una directory creata appositamente all'interno della memoria dello smartphone adibita esclusivamente alla raccolta dei dati scaricati dal Raspberry.

Al suo interno si trovano tutti i video scaricati e per ognuno di esso, un file testuale contenente le informazioni aggiuntive che non possono essere ricavate dall'applicazione: timestamp della data di collisione e i dati di accelerazione. La data dell'urto è infatti impostata dal Raspberry e non corrisponde alla data di creazione del video sullo smartphone. La lista dei video mostrata all'utente viene popolata con le informazioni presenti all'interno di questa directory: ad ogni avvio viene effettuato un matching tra il nome del video e il quello del file di informazioni per associarle correttamente. In figura viene mostrata una schermata dell'applicazione con la lista dei video presenti.

Come è possibile notare, sono presenti ulteriori elementi grafici associati ad ogni video. Sulla sinistra è stata aggiunta un'immagine contenente un frame del relativo video per consentire all'utente di identificarlo immediatamente, evitando di dover

ricordare il nome o la data in cui è stato creato. Il simbolo a forma di stella indica invece che un video è stato impostato come preferito: un'ulteriore agevolazione in favore dell'utente che in questo modo può forzare i suoi preferiti ad essere sempre in testa alla lista per un accesso più rapido.

3.7.5 Lista dei video sul Raspberry

La lista dei video è una delle informazioni più importanti che il Raspberry fornisce all'utente e rappresenta l'insieme delle collisioni rilevate e registrate. Dopo aver completato l'operazione iniziale di configurazione con il Raspberry, l'applicazione è in grado di accedervi ed ottenere tutte le informazioni memorizzate al suo interno. La figura mostra una schermata dell'applicazione in cui è possibile vedere la lista completa dei video disponibili.

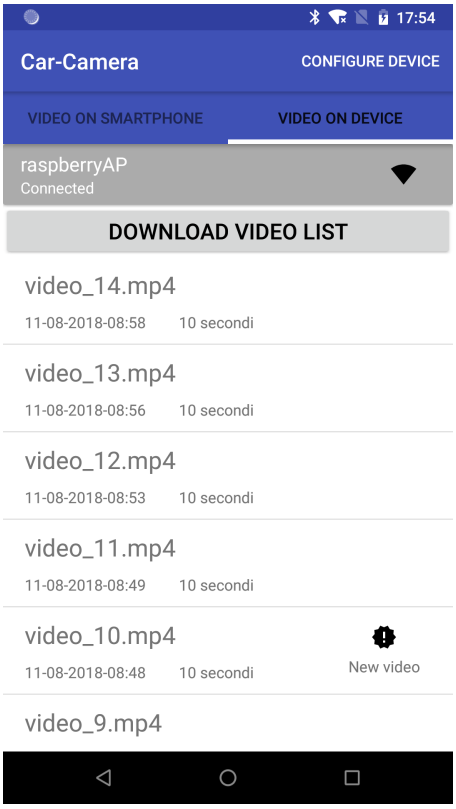

Figura 3.4. Lista dei video memorizzati sul Raspberry

Barra di stato

Nella figura 3.4 è possibile notare una barra di stato che ha lo scopo di segnalare alcune informazioni utili all'utente:

- Mostrare la potenza del segnale della rete wifi del Raspberry. Questa informazione risulta superflua quando i due dispositivi sono connessi, poichè è un dato già presente nella barra di stato di Android. La decisione di aggiungerla è stata presa in base a come il sistema operativo gestisce la connessioni a reti

che non forniscono accesso ad internet. Infatti Android tende ad interromperle e utilizzare la rete dello smartphone oppure altre reti wifi conosciute. La possibilità di poter visualizzare qui la potenza del segnale, evita all'utente di:

  - Provare a collegare lo smartphone alla rete del Raspberry, senza essere sicuro che sia ad un distanza sufficiente, con il rischio che la connessione non vada a buon fine.
  - Utilizzare la liste delle reti wifi offerta da Android, che mostra anche la potenza del segnale rispetto ad ogni rete, prima di collegarsi.

- Comunicare all'utente lo stato della connessione con la rete del Raspberry:

  - Connesso: l'applicazione è correttamente connessa alla rete del dispositivo ed è possibile instaurare una comunicazione.
  - Disconnesso: è necessario collegarsi al Wi-Fi del Raspberry per iniziare la trasmissione dei dati.
  - Fuori portata: l'applicazione non rileva la presenza del Raspberry all'interno del raggio di azione, i due dispositivi sono troppo distanti.

Richiedere la lista dei video

Il pulsante sotto la barra di stato consente all'utente di ricevere la lista aggiornata dei video sul Raspberry, inviando la specifica richiesta discussa nella sezione 3.4.4. L'applicazione riceve una stringa, serializzata in json, contenente la lista dei video e per ognuno il Raspberry invia:

- Il nome del video
- La data della collisione
- La durata del file

Oltre a mostrare tutte queste informazioni all'utente, l'applicazione provvede anche ad aggiungere un'icona che segnala se quel particolare video è stato già scaricato o meno, mettendolo in evidenza ed etichettandolo come nuovo.

Questo evento ha anche un effetto sul funzionamento del service, adibito alla ricezione di pacchetti Bluetooth contenenti il numero dei nuovi video da notificare. Come discusso nella sezione 3.7.1, viene utilizzata una variabile per distinguere una notifica già comunicata all'utente da una nuova, per evitare di segnalare la presenza continua dello stesso video. Dopo aver ottenuto la lista dei video, lo stato del sistema cambia:

- L'applicazione ha fornito all'utente le informazioni più recenti disponibili sul Raspberry e questo abilita la ricezione di nuove notifiche. Il valore dei video da notificare viene azzerato per consentire al service di avvisare l'utente collisioni che avverrano da questo momento in poi.

- Il Raspberry ha inviato della lista dei video che è una prova dell'interazione dell'utente con l'applicazione: questo permette la terminazione della fase di advertising, in quanto conferma del fatto che sono state inviati i dati più recenti. Infatti il Raspberry non è più in possesso di video non sono stati ancora richiesti e dovrebbe inviare un valore pari a zero all'interno dei pacchetti di advertising.

Scaricare un video

Dopo aver ricevuto la lista dei video, l'utente ha la possibilità di selezionare quelli di suo interesse ed iniziare il download. Il video da scaricare è identificato dal proprio nome, univoco all'interno del sistema, che l'applicazione provvede ad inviare al Raspberry: questa funzione corrisponde al terzo punto del protocollo testuale, descritto nella sezione 3.4.4. In seguito a questo richiesta, il Raspberry recupera il file multimediale e lo invia all'applicazione. Anche il vettore delle accelerazioni viene trasmesso, ma serializzato in json, per consentire all'applicazione di deserializzarlo direttamente in una struttura dati. Le rimanenti informazioni, invece, non vengono inviate perchè già presenti all'interno della lista dei video, precedentemente trasmessa e sono già in possesso dell'applicazione.

Per quanto riguarda l'effettiva connessione con il server sul Raspberry, è stato necessario implementare simmetricamente le stesse funzioni discusse nella sezione 3.4.4:

- Possibilità di scaricare più video contemporaneamente, così da poter sfruttare il multithreading presente sul server e non dover attendere la fine di una ricezione, prima di iniziarne un'altra.

- Creazione di un canale di trasmissione sicuro per assicurare l'autenticità e l'integrità dei dati, utilizzando il protocollo crittografico TLS.

Il trasferimento parallelo è gestito da un thread pool, da cui viene estratto un nuovo thread per ogni richiesta dell'utente e si occupa di:

- Controllare che il video non sia stato già scaricato, prima di inviare il relativo nome al Raspberry, verificando l'esistenza sia del file multimediale che del file testuale contenente le informazioni. Nel caso in cui esistano entrambi, viene notificato l'utente e la ricezione interrotta perchè superflua.

- Inviare il nome del video da scaricare e ricevere sia il file che i dati di accelerazione. In questa fase vengono effettuate alcune modifiche all'interfaccia grafica per fornire un riscontro all'utente sullo stato di avanzamento del processo. In primo luogo viene cambiata la schermata di visualizzazione, mostrando la lista dei video già presenti sul dispositivo, a cui viene aggiunto quello richiesto. Sull'item della lista relativo al video in questione appare una progress bar che segna lo stato di avanzamento dello scaricamento in percentuale.

- Ricevere i dati di accelerazione del veicolo al termine dello scaricamento del video. Questi dati non saranno mostrati all'utente per il momento, ma memorizzati sullo smartphone.

- Creare il file testuale con le informazioni della collisione, compresi i valori di accelerazione, all'interno della stessa directory del video.

Al termine di queste fasi, la lista dei video sarà aggiornata con il nuovo appena scaricato e l'utente ha la possibilità di visionarlo.

Client TLS

La trasmissione sicura dei dati è stata realizzata, anche in questo caso, utilizzando una specifica class di Android che consente la creazione del canale sicuro e cifrato per la comunicazione. Come precedentemente detto, il server sul Raspberry non richiede l'autenticazione del client che quindi non utilizza alcun certificato X.509 per dimostrare la sua identità. D'altro canto, l'autenticazione del server è sempre obbligatoria e durante l'handshake, provvede sempre ad inviarlo. Il problema a questo punto riguarda proprio la verifica dell'identità del server, infatti il suo certificato è self-signed e perciò senza la firma di nessuna trusted CA. I socket possono essere istruiti per verificare la validità di un certificato ed è possibile impostare uno o più server fidati, che se figurano durante la verifica, segnano il certificato come valido e l'handshake può continuare.

In questo contesto è possibile agire sull'implementazione del socket per fare in modo che accetti qualsiasi tipo di certificato, anche senza la firma di alcuna CA: ovviamente è una soluzione valida solo in questo contento in cui non si dispone di certificati validi, perchè espone la comunicazione ad attacchi da parte di terzi.

3.8 Visualizzazione dei dati

Viene infine presentata la schermata di maggiore interesse per l'utente, che gli permette di visualizzare le informazioni riguardo una collisione avvenuta: per accedervi è sufficiente selezionare uno specifico video all'interno della lista di quelli presenti sul dispositivo. La schermata ha il solo scopo di fornire una rappresentazione grafica dei dati che riguardano il video, specialmente dei valori di accelerazione che finora sono stati presentati come una semplice sequenza di numeri.

Tutti questi dati vengono trasmessi internamente all'applicazione a seguito della selezione del video, tramite un Intent. In Android un Intent è un concetto astratto che descrive una certa operazione da eseguire, che può essere effettuare una chiamata, inviare una mail, fare una ricerca sul web o simili. Gli Intent possono essere anche utilizzati per avviare una nuova activity, come nel caso in analisi, ed inoltre è possibile inserire dei dati al suo interno, in modo che il destinatario possa riceverli ed usarli per i propri scopi. L'oggetto che modella il video della lista viene serializzato ed inviato all'activity che può ricostruirlo e accedere alle informazioni in esso contenute.

Media Player

La visualizzazione del video dell'urto è affidata al Media Player di default di Android, che include il supporto per la visualizzazione di numerosi formati di file multimediali. E' possibile riprodurre contenuti memorizzati localmente sul dispositivo oppure proveniente ad fonti esterne, come lo streaming da internet. Nel

caso in analisi, il video da riprodurre si trova nella memoria dello smartphone ed è sufficiente recuperare la sua posizione esatta, formata dalla directory utilizzata dall'applicazione a cui viene aggiunto il nome del file, ottenuto dall'Intent. Dopo aver fornito al Media Player la sorgente del video da riprodurre, è necessario indicargli dove mostrarlo. Infatti, Il Media Player richiede, una superficie o Surface, come viene definita in Android, con ha lo scopo di consumare, ovvero mostrare sull'interfaccia grafica i frame del video da riprodurre. Tra le diverse Surface su cui è possibile riprodurre il video è stata utilizzata una GLSurfaceView che fornisce alcune funzionalità aggiuntive:

- Gestisce una surface su cui è possibile usare il rendering OpenGL.
- Consente l'uso di un oggetto custom Renderer per manipolare la surface, sia in modalità on-demand che continua.
- Diversamente dalle altre View, il rendering è effettuato da un thread secondario, per evitare che le performance interferiscano con le operazioni del thread dedicato all'interfaccia grafica.

Media Controller

Il Media Player, da solo, offre solo la possibilità di visualizzare il video, senza però poterlo manipolare o controllare e viene infatti spesso affiancato da un altro componente, che ne arricchisce il funzionamento. Il Media Controller è una semplice View, un oggetto grafico di Android rappresentato come una barra, posizionato nella parte bassa del Media Player e contiene numerosi controlli per gestire il file multimediale durante la sua riproduzione. Normalmente il Media Controller è dotato dei più comuni pulsanti usati per la manipolazione dei video, come play/pausa, i tasti di forward e rewind per avanzare o tornare indietro di alcuni secondi e riporta anche l'istante corrente di riproduzione e la durata totale del video.

Inoltre mette a disposizione una progress bar che mostra lo stato di avanzamento e può essere usata per accedere velocemente ad un istante preciso del video. Il Media Controller si integra perfettamente con il Media Player ed effettua automaticamente la sincronizzazione del video con i suoi controlli, compresa la durata e lo stato della progress bar.

OpenGL

OpenGraphics**L**ibrary è una libreria grafica cross-platform che fornisce delle API standard per la realizzazione di contenuti grafici, specialmente tridimensionali. Il grande vantaggio di questi API è la loro interazione con la GPU del dispositivo. Infatti grazie all'elevato grado di parallelismo che possono fornire le GPU attuali, è possibile ottenere un notevole aumento di prestazioni rispetto all'esecuzione sulla CPU. Android utilizza in particolare le OpenGL ES API che forniscono un sotto insieme delle funzioni di OpenGL, ottimizzato per i dispositivi mobile ed embedded.

La GLSurfaceView è la surface su cui vengono mostrati i frame prodotti dal Media Player, ma con l'utilizzo di un Renderer, è possibile effettuare il rendering OpenGL. Il Renderer è, nello specifico, una generica interfaccia, che si occupa di effettuare le chiamate OpenGL per il rendering sui singoli frame ed inoltre fornisce alcune callback riguardanti il ciclo di vita della surface a cui è collegata:

- onSurfaceCreated: è la prima funzione chiamata, corrisponde all'istante di creazione della surface. Le operazioni che vengono svolte solitamente al suo interno riguardano l'inizializzazione di risorse e la configurazione dell'ambiente OpenGL, per poter utilizzare le sue funzioni successivamente. Le Textures, ad esempio dovrebbero essere create all'interno di questa funzione.

- onSurfaceChanged: viene chiamata ogni volta che cambia la dimensione della surface su cui viene effettuato il rendering.

- onDrawFrame: è la funzione responsabile di disegnare sui frame, appena sono disponibili.

Texture

Una Texture è un oggetto di OpenGL che contiene i dati di un'immagine che possono essere manipolati dalla libreria grafica. Per fare in modo che i frame prodotti dal Media Player potessero essere utilizzati da OpenGL, è stata creata una Surface usata come superficie per la riproduzione del video, sui cui viene disegnata la texture dopo il rendering di OpenGL. I dati che vengono caricati sulla texture possono essere utilizzati dalla pipeline grafica che si occupa di eseguire una serie di operazione per generare l'immagine finale. Alcuni passi della pipeline sono fissi, invece per eseguirne altri è necessario compilare e caricare dei programmi durante la configurazione del renderer. E' questo il caso degli shader: programmi scritti in un particolare linguaggio che vengono eseguiti all'interno della GPU.

Shaders

Gli shader sono dei programmi scritti nel linguaggio GLSL. OpenGL ES 2.0 supporta due tipi di shader programmabili all'interno della pipeline:

- Vertex shader: il suo compito è calcolare la posizione di finale di ogni vertice partendo dallo spazio del modello allo spazio dell'immagine da rappresentare.

- Fragment shader: prende in input la posizione di ogni pixel, calcolata dal vertex shader a cui assegna il colore. E' lo shader in grado di effettuare le operazioni più sofisticate perchè viene eseguito in maniera indipendente per ogni pixel che deve essere disegnato.

Gli shader possono interagire con le parti fisse della pipeline grazie all'accesso a speciali variabili di input e output. Il vertex shader, ad esempio, deve impostare il valore di gl_Position con le coordinate del punto corrispondente al vertex nello spazio dell'immagine. Allo stesso modo, il fragment shader riceve la posizione del pixel all'interno di gl_FragCoord e deve specificare il suo colore in gl_FragColor.

L'utilizzo di una GLSurfaceView ed la conseguente manipolazione del video con OpenGL si è rivelata necessaria a causa di una delle caratteristiche del video, non ancora esposta. Come detto, la video camera hanno un campo visivo molto ampio per consentire la registrazione dell'intero ambiente circostante. Le video camere hanno una lente che viene definita fisheye.

Fisheye

Le lenti fisheye sono delle lenti ultra wide-angle che permettono la registrazione di una scena con un campo visivo estremamente maggiore rispetto alle normali lenti. Sono molto utilizzate dai fotografi per la creazione di effetti speciali o per riprendere scene molto ampie, come un paesaggio esterno. Il risultato finale è un'immagine con ha un angolo di vista di circa 180°. Il rovescio della medaglia sta in come l'immagine viene mostrata: una lente fisheye per ottenere questo risultato, non può che distorcerla. In particolare, la distorsione è circolare ed direttamente proporzionale alla distanza dal centro. In figura è mostrato un esempio di immagine scattata con la video camera.

Questa caratteristica della video camera è senza dubbio utile per la registrazione del video, ma la distorsione creata dalla lente è tutt'altro che trascurabile. Per questo motivo è stato usato il fragment shader per correggere il video durante la sua riproduzione.

Figura 3.5. Immagine distorta

In figura 3.5 è possibile vedere una foto scattata con la video camera utilizzata. La parte centrale dell'immagine è effettivamente priva di distorsione o minima, infatti sembra scattata con una normale lente. Le zone visibile negli angoli sono invece completamente deformate ed è possibile notare l'aumentare della distorsione, proporzionale alla distanza dal centro dell'immagine.

La distorsione provocata dalle lenti fisheye viene spesso definita barrel distortion [25] perchè l'immagine sembra mappata su una sfera o appunto un barile. E' questo il modo in cui la lente distorce l'immagine, provando a far entrare ciò che inquadra con il suo angolo di vista molto ampio, in uno spazio finito. La distorsione radiale è il principale effetto della barrel distortion, su cui è possibile intervenire, utilizzando modelli matematici che esprimono una relazione tra un punto dell'immagine distorto da uno non distorto.

Il modello di distorsione di Brown-Conrady [26] modella la distorsione di un'immagine in base sia alla distorsione radiale che tangenziale. Nella formula [27] è riportata solo la componente della distorsione radiale.

$$x_u = x_d + (x_d - x_c)(K_1 * r_d^2 + K_2 * r_d^4)$$
$$y_u = y_d + (y_d - y_c)(K_1 * r_d^2 + K_2 * r_d^4)$$

- (x_u, y_u): coordinate di un punto non distorto
- (x_d, y_d): coordinate di un punto distorto
- (x_c, y_c): coordinate del centro di distorsione
- (r_d): $\sqrt{(x_d - x_c)^2 + (y_d - y_c)^2}$ rappresenta la distanza tra il centro di distorsione ed un punto dell'immagine distorta
- K_n: coefficiente n-esimo della distorsione radiale

La distorsione radiale del modello è dominata principalmente delle prime due componenti, le altre sono state omesse dalla formula. Inoltre, il modello divisionale [28] è spesso preferibile per la correzione radiale, a quello polinomiale di ordine pari, proposto da Brown-Conrady [29].

$$x_u = x_c + \frac{x_d - x_c}{(1 + K_1 * r_d^2 + K_2 * r_d^4)}$$

$$y_u = y_c + \frac{y_d - y_c}{(1 + K_1 * r_d^2 + K_2 * r_d^4)}$$

Sulla base dell'ultima formula, è stato calcolato il fattore di correzione tra il raggio distorto dell'immagine originale ed il raggio corretto [30].

$$r_c = r_d(1 + K_1 * r_d^2 + K_2 * r_d^4)$$

- r_c : valore del raggio corretto dopo aver applicato i coefficienti di correzione radiale K1 e K2

Dopo aver calcolato il raggio corretto sono state usate alcune formule di trigonometria per ottenere le coordinate del nuovo punto. Con la prima formula è stato calcolato l'angolo formato dalle coordinate distorte dispetto al centro di distorsione:

$$theta = \arctan(x_d - x_c, y_d - y_c)$$

Infine le coordinati dei punti corretti vengono messe in relazione con l'angolo ed il valore del raggio corretto:

$$x_u = \cos(theta) * r_c$$

$$y_u = \sin(theta) * r_c$$

I parametri K1 e K2 rappresentano i coefficienti di distorsione radiale e determinano il tipo di distorsione e la sua entità. La barrel distortion è stata corretta con una distorsione definita pinchusion, diametralmente opposta alla barrel, in cui sembra che i punti dell'immagine vadano a confluire verso il suo centro. Sulla base del modello proposto, il segno dei coefficienti di distorsione radiale determinano il tipo di distorsione finale: valori negativi di K1 provocano una pinchusion distorsion. I valori di K1 e K2 sono stati calcolati con un programma di test, utilizzando un'immagine di prova catturata dalla video camera, variandoli in contemporanea, fino ad ottenere la correzione desiderata.

Il calcolo delle coordinate non distorte viene effettuato per ogni punto della texture utilizzata da OpenGL, per merito del fragment shader che viene eseguito per ogni pixel. Per effettuare la correzione è stata utilizzata una funzione di GLSL che permette di ottenere ogni texel della texture in esame. Un texel è semplicemente il colore che il fragment shader ha associato ad ogni pixel della texture. Le operazioni per la correzione dell'immagine all'interno del fragment shader sono le seguenti:

- Ricezione delle coordinate della texture dal vertex shader all'interno di vettore bidimensionale uv.
- Correzione dei nuovi valori di uv utilizzando il modello sopra descritto.
- Calcolo di ogni texel della texture corrispondente alle coordinate corrette.
- Restituzione dei nuovi colori all'interno della variabile gl_FragColor.

Quello che viene fatto per correggere la distorsione è un cambio dei colori assegnato dal fragment shader, che vengono calcolati a partire dalle coordinate corrette. Ogni texel viene spostato dalle sue originali coordinate, a quelle calcolate dopo l'applicazione del modello matematico, mostrando all'utente l'immagine priva della distorsione introdotta dalle lenti fisheye.

Figura 3.6. Immagine corretta

L'immagine in figura 3.6 è stata ottenuta dopo aver applicato le correzioni appena descritte alla foto distorta scattata con la video camera.

Grafico delle accelerazioni

I dati di accelerazione sono invece stati riportati su un grafico disegnato utilizzando una custom view, un componente di Android che permette la modellazione e personalizzazione dell'interfaccia grafica. Il grafico è un sistema di assi cartesiani con i valori di accelerazioni sull'asse delle ordinate ed il tempo sull'asse delle ascisse. Le accelerazioni hanno in realtà una sola coordinata, che corrisponde al totale delle accelerazioni rilevato sui 3 assi, ma non hanno un riferimento all'istante del video in cui sono stati calcolati. La loro posizione sull'asse delle ascisse viene determinata in base alla dimensione della custom view ed alla frequenza con cui vengono letti dall'accelerometro, che indica il numero totale di valori memorizzati dal Raspberry, per ogni secondo di video. Il grafico viene infine disegnato unendo ogni punto con il successivo, in modo da mostrare l'andamento delle accelerazioni.

Visualizzare completamente il grafico, però, potrebbe essere poco interessante da parte dell'utente, che probabilmente vuole poter vedere in quale istante preciso del video è stata registrata l'accelerazione maggiore, corrispondente all'urto ricevuto dal proprio veicolo. Si è quindi pensato di rendere il grafico dinamico facendo il modo che il suo andamento vada di pari passo con il video, sincronizzando ogni secondo di riproduzione con i rispettivi valori di accelerazione. Le basi per realizzare questa funzionalità sono già presenti in quanto il Raspberry inizia contemporaneamente sia la memorizzazione sia dei frame che delle accelerazioni. In maniera implicita è già possibile determinare, conoscendo l'istante della collisione, quali sono i relativi valori di accelerazione, moltiplicando il numero dei secondi trascorsi per la frequenza di lettura dei dati dal sensore. Questo è il punto di partenza per

la realizzazione di questa funzionalità, ovvero conoscere la relazione tra il numero dei secondi del video ed il numero totale del vettore delle accelerazioni.

Per rendere il video dinamico, si è semplicemente pensato di nascondere i valori di accelerazione successivi all'istante corrente di riproduzione del video: ad ogni secondo di avanzamento corrisponde anche un avanzamento del grafico, che alla fine del video sarà completo. Quello che manca a questo punto, è conoscere in ogni istante lo stato di riproduzione del video e comunicarlo alla custom view, così che possa aggiornare il grafico.

Sincronizzare il grafico e video

L'unico componente che può essere utilizzato in questa situazione è proprio il Media Controller ed in particolare la sua progress bar, che viene aggiornata automaticamente dal Media Player. Il grafico deve rispondere sia alla naturale riproduzione del video che alla modifica della progress bar da parte dell'utente, provocando un cambiamento rapido della visualizzazione del video. La differenza tra queste due condizioni è determinata dalla visibilità della progress bar: il Media Controller è posizionato nella parte bassa del Media Player e per non limitarne la visuale, diventa visibile solo in seguito ad un'interazione da parte dell'utente e per pochi secondi. Purtroppo, quando non visibile il Media Controller, non c'è la possibilità di ottenere il valore corrente della progress bar e non si potrebbe aggiornare il grafico.

La soluzione più semplice sarebbe stata la modifica del tempo di visibilità del Media Controller, impostandolo allo stesso valore della durata del video, così da renderla sempre visibile e di conseguenza non ci sarebbe stato il problema del mancato aggiornamento del suo stato di avanzamento. Questo avrebbe portato ad una riduzione delle visuale del video, proprio a causa della posizione del Media Controller, che ricopre ne la parte inferiore ed è stata quindi scartata.

Si è scelto infine di utilizzare un approccio ibrido, ricavando lo stato di avanzamento del video dalla progress bar solo quando è visibile; in caso contrario, si recupera l'istante corrente del video dal Media Player che lo restituisce in ogni condizione.

Aggiornare il grafico dal Media Player

L'aggiornamento del grafico dal Media Player deve essere effettuato in maniera continua, controllando l'istante di avanzamento e comunicandolo alla custom view. La lettura continua è necessaria per evitare che il grafico rimanga fermo a causa della mancanza dell'aggiornamento dal Media Player. Per questo motivo, l'operazione è stata delegata ad un thread secondario, che ad intervalli regolari e frequenti, calcola il progresso corrente del video e aggiorna il grafico. Ci sono però delle condizioni in cui il thread deve essere messo in attesa:

- Quando il video viene messo in pausa il thread si mette in attesa, evitando di aggiornare continuando la custom view con lo stesso valore. La visualizzazione del grafico non cambia, ma la custom view effettua lo stesso i calcoli necessari per modificarlo.
- Durante una modifica della progress bar da parte dell'utente. In questo caso è già la progress bar che provvede ad aggiornare il grafico e non è necessario che venga fatto lo stesso anche dal Media Player.

Aggiornare il grafico dalla progress bar

In Android è possibile consultare lo stato della progress bar, impostando un *OnSeekBarChangeListener*, una semplice interfaccia che comunica i principali cambiamenti di stato. Le funzioni dell'interfaccia sono:

- *onProgressChanged(SeekBar seekBar, int progress, boolean fromUser)*: chiamata ogni volta che il valore della progress bar cambia, comunicando il nuovo valore all'interno della variabile *progress*. E' anche possibile distinguere se l'avanzamento è stato provocato dalla naturale riproduzione del video o dall'interno dell'utente, consultando *fromUser*.
- *onStartTrackingTouch(SeekBar seekBar)*: questa funzione viene chiamata ogni volta che l'utente effettua una gesture sulla progress bar.
- *onStopTrackingTouch(SeekBar seekBar)*: viene invece chiamata ogni volta che l'utente termina una gesture.

All'interno di queste funzioni viene effettuato sia l'aggiornamento del grafico che la sincronizzazione con il thread che riceve i valori dal Media Player, per fare in modo che comunichino con la custom view in mutua esclusione. Come detto prima, la progress bar diventa visibile solo in seguito all'intervento dell'utente, che effettua una gesture sul Media Player, mostrando il Media Controller: da questo momento in poi, è possibile ricevere le informazioni all'interno delle callback sopra descritte. Le funzioni onStartTrackingTouch e onStopTrackingTouch possono essere utilizzate per ricavare l'intervallo di tempo in cui l'utente sta effettivamente muovendo la progress bar ed il thread deve quindi essere messo in attesa, per evitare di aggiornare contemporaneamente il grafico.

La sincronizzazione avviene tramite una condition variable su cui il thread attende durante l'utilizzo della progress bar. Appena l'utente inizia a muovere la progress bar, in *onStartTrackingTouch* viene cambiato il valore di una variabile che provoca la messa in attesa del thread. Viceversa, in *onStopTrackingTouch* viene notificata la condition variable ed il thread ricomincia ad aggiornare il valore del grafico. Tra la chiamata di queste due funzioni sappiamo con certezza che il thread non sta effettuando alcuna operazione e che l'utente sta effettivamente interagendo con la progress bar. Di conseguenza, il Media Controller è sicuramente visibile ed è possibile aggiornare il grafico con il valore che la progress bar restituisce in *onProgressChanged*. Inoltre, durante l'utilizzo della progress bar da parte dell'utente, viene anche aggiornato l'avanzamento del video, per preservare la coerenza con l'istante di visualizzazione del grafico.

Figura 3.7. Visualizzazione del video e del grafico delle accelerazioni

In figura 3.7 è stata riportata la schermata dello smartphone che mostra il MediaPlayer durante la riproduzione del video e la custom view, su cui viene disegnato il grafico delle accelerazioni.

Il video e il grafico delle accelerazioni consentono all'utente di ricostruire la dinamica dell'accaduto ed avere una visione più chiara dell'urto ricevuto dal proprio veicolo. La visualizzazione separata di queste due informazioni non avrebbe portato allo stesso risultato. I dati prodotti dalla video camera sono senza dubbio i più rilevanti perchè offrono un riscontro visivo ed una prova inconfutabile dell'accaduto. Inoltre sono in grado di identificare il responsabile della collisione, mostrando la targa del sua veicolo oppure possono essere essere utilizzati dalla compagnia assicurativa contro un tentativo di frode da parte del proprietario della vettura. Senza i dati di accelerazione, però, sono meno significativi in quanto rendono impossibile quantificare l'entità del danno e ciò potrebbe portare ad un risarcimento non adeguato. Il grafico delle accelerazioni, allo stesso modo, è privo di significato se non associato al rispettivo video.

Sono queste le motivazioni che hanno portato alla realizzazione dell'interfaccia grafica comprensiva di tutte le principali informazioni raccolte dal sistema, riguardanti una certa collisione. La possibilità di visualizzare il video e le accelerazioni nello stesso istante di tempo semplifica notevolmente la comprensione dell'incidente. La sincronizzazione tra il Media Player ed il grafico agevola l'utente nella visione del video perchè gli permette di avanzare nella riproduzione del video e soffermarsi sui momenti salienti, senza doversi preoccupare anche di cambiare la visualizzazione delle accelerazioni, l'applicazione aggiornerà automaticamente il grafico.

Video preferito

Nella figura si possono notare altre due icone che corrispondono ad altrettante funzionalità che sono a disposizione dell'utente. La prima offre la possibilità di impostare un video come preferito così da accedervi più velocemente, grazie al riposizionamento all'interno della lista. Il cambiamento di questa opzione avviene localmente allo smartphone, perchè ci si riferisce ai video che sono stati precedentemente scaricati. Come detto in precedenza, tutte le informazioni riguardanti i video sono memorizzate all'interno del file testuale associato, presente nella stessa directory. Tra queste c'è anche un campo che riporta se un video è stato impostato come preferito e viene letto in fase di creazione della lista per eseguire l'ordinamento. La preferenza viene usata in combinazione con il timestamp affinchè negli elementi superiori della lista ci siano tutti i video preferiti e se più di uno, sono presentati in ordine temporale, con i più recenti in alto.

Condividere un video

L'altra funzionalità offerta all'utente è la possibilità di condividere i video con i propri contatti. La scelta di introdurre in questa schermata questa opzione è una diretta conseguenza della directory scelta per la memorizzazione dei video. Come accennato in precedenza, i video non vengono salvati nella galleria dello smartphone, ma in una directory creata appositamente all'interno del file system. Il pulsante di condivisione inserito in questa schermata, permette all'utente di eseguire velocemente l'operazione, senza doversi preoccupare di conoscere la sua esatta locazione. Questo è sicuramente un vantaggio dal punto di visto dell'usabilità dell'applicazione e della sua efficienza, perchè consente di eseguire in maniera rapida le principali operazioni sui video.

Le diverse possibilità di condivisione vengono offerte direttamente da Android, mostrando tutte le applicazioni installate sullo smartphone che permettono l'invio dei file multimediali. Alcuni esempi sono le applicazioni di social messagging, la posta elettronica e la condivisione tramite Bluetooth. Questa funzione può essere sicuramente di interesse per quanto riguarda la comunicazione con la compagnia assicurativa, a cui si può inviare la prova del danno ricevuto al proprio veicolo.

3.9 Misurazioni

In questa ultima sezione vengono mostrati i dettagli di alcune misurazioni effettuate durante l'utilizzo del sistema. Lo scopo di questa analisi è fornire un'idea generale della quantità di dati necessaria per il corretto funzionamento del sistema. Le misurazioni non riguardano un dispositivo in particolare, ma tutti i diversi componenti del sistema che, in base alle proprie caratteristiche hardware, hanno bisogno di una certa quantità di memoria. Il Raspberry è sicuramente il dispositivo più sensibile per l'analisi effettuata, in quanto deve provvedere a fornire le risorse sia alla video camera che all'accelerometro. Infatti, la scelta del Raspberry come componente principale del sistema è stata preceduta da una valutazione sulle sue caratteristiche hardware, per assicurarsi che fosse idoneo allo scopo.

In particolare, la video camera è il dispositivo che ha richiesto le valutazioni più approfondite, perchè la dimensione dei frame generati dipende molto dalla

risoluzione e dal formato di compressione supportati. Inoltre, va considerato anche il frame rate che determina il numero di frame necessari per descrivere un secondo di video.

Memoria occupata dal buffer circolare

La prima misurazione riguarda la quantità di memoria occupata dal buffer circolare che memorizza i frame ricevuti dalla video camera. La struttura dati viene mantenuta costantemente in memoria per consentire al Raspberry di recuperare i dati in seguito ad una collisione, ma la sua dimensione può essere considerata circa costante, perchè i dati vengono periodicamente sovrascritti. La misurazione è stata effettuata con un programma di test che riceve i dati dalla video camera e calcola la quantità di memoria occupata dal buffer circolare ogni volta che viene riempito completamente. Di seguito vengono elencati i parametri utilizzati per registrare il video e per dimensionare il buffer:

- Dimensione del buffer: 100 posizioni
- Risoluzione video camera: 1920 x 1280
- Formato di compressione: H.264
- Frame per secondo: 30

MB totali	KB per frame
77.502	26.454
77.168	26.34
77.268	26.374
75.560	25.791
77.025	26.291
77.401	26.419
77.505	26.455
77.482	26.447
77.319	26.391
77.435	26.431
77.114	26.321
77.132	26.328
75.671	25.829
77.413	26.423
77.579	26.480

Tabella 3.4. Memoria occupata dal buffer circolare

Nella tabella 3.4 sono mostrati i risultati ottenuti, su un totale di 15 misurazioni. Le prove sono state effettuate in diverse condizioni, variando la luminosità dell'ambiente e riprendendo sia soggetti fermi che in movimento. Come è possibile notare, la dimensione media del buffer in memoria è di circa 77 MB, per un totale di 100 secondi di video memorizzati.

Considerando l'elevata risoluzione ed il frame rate utilizzando, i valori di memoria necessari riportati in tabella sono bassi rispetto alla qualità del video che garantiscono. Il merito è esclusivamente della compressione H.264 della video camera, capace di descrivere un singolo frame in appena 26 kB. Sulla base di questi risultati, è evidente che la dimensione della memoria del Raspberry (1 GB) è più che sufficiente per memorizzare queste informazioni; questo offre la possibilità di aumentare la dimensione del buffer e della durata del video da creare.

I parametri per realizzare queste misurazioni sono gli stessi utilizzati nella reale implementazione del sistema. La dimensione del buffer è stata stabilita ipotizzando la creazione di video di 1 minuto. Dal calcolo della memoria necessaria è stata esclusa la dimensione del buffer che memorizza i valori di accelerazione perchè trascurabile rispetto alla struttura dati per il salvataggio dei frame. Infatti, il quantitativo di memoria necessario è di almeno un ordine di grandezza inferiore. Ipotizzando di mantenere in memoria 100 secondi di dati e una di leggere dall'accelerometro 10 valori al secondo:

$$dim_{buffer} = 100 * 10 * sizeof(float) \approx 4\,\text{kB}$$

Video storage

Dopo aver analizzato il quantitativo di memoria RAM necessari per la memorizzazione temporanea dei dati all'interno del sistema, è stata effettuata una seconda analisi che riporta la dimensione dei video creati in seguito ad una collisione. Questi dati sono rilevanti sopratutto dal punto di vista del Raspberry che deve mantenere su sulla SD card tutti i video generati e di conseguenza deve essere dimensionata adeguatamente. Inoltre, anche la dimensione dei video ha un effetto anche sull'applicazione Android, non solo per la memorizzazione all'interno dello smartphone, ma anche perchè determina il tempo necessario per la loro trasmissione. I risultati ottenuti sono mostrati nella tabella 3.5.

MB su disco	**Durata del video**
74.8	92 secondi
58.5	72 secondi
74.6	92 secondi
74.7	92 secondi
27.7	34 secondi
50.4	62 secondi
48.8	60 secondi
35.8	44 secondi
50.4	62 secondi
50.3	62 secondi

Tabella 3.5. Dimensione dei video

La dimensione dei video su disco è prevedibile conoscendo la memoria occupata dai frame all'interno del buffer, mostrata nella precedente analisi. Per la creazione del video, infatti, la libreria FFmpeg si occupa di copiare i singoli frame e scriverli

all'interno del video senza ulteriori operazioni. Nonostante ciò, si è preferito effettuare ugualmente la presente analisi, specialmente per mostrare la dimensione dei video di differenti durate.

In generale, lo spazio necessario per l'archiviazione è contenuto e questo permette al Raspberry di memorizzare un elevato numero di video prima di riempire il disco (per l'implementazione è stata utilizzata una SD card da 4 GB).

Capitolo 4

Conclusioni

In questo lavoro di tesi è stato effettuato uno studio sulla fattibilità del sistema proposto, al fine di valutarne l'utilità in un contesto reale. Le problematiche a cui si è cercato di porre rimedio riguardano i proprietari dei veicoli in primo luogo, ma anche le compagnie assicurative possono trarne vantaggio. Monitorare il proprio veicolo durante una sosta ed avere una prova di un'eventuale collisione è la funzionalità principale del sistema ed i proprietari possono avere maggiori probabilità di ottenere un risarcimento dall'assicurazione del responsabile dell'urto. Invece, le compagnie assicurative possono utilizzare il dispositivo come deterrente contro le frodi.

Gli strumenti impiegati sono solo un esempio di come può essere realizzato il sistema che non presenta alcun vincolo di tipo software o hardware. Nonostante ciò, la scelta dei componenti è stata valutata attentamente, ponendo particolare enfasi sulle loro caratteristiche hardware per assicurarsi di avere a disposizione le necessarie risorse computazionali per la creazione dei video e il salvataggio dei valori di accelerazione.

L'implementazione del sistema è stata seguita da una fase di analisi delle prestazioni che ha permesso la valutazione dell'efficienza del sistema durante la fase di elaborazione dei dati e della loro trasmissione all'applicazione. In generale, l'implementazione realizzata con i dispositivi descritti è stata ritenuta soddisfacente sotto diversi punti di vista, specialmente dal punto di vista dei sensori esterni che hanno permesso l'acquisizione di dati precisi e rigorosi. L'accelerometro, con la sua elevata sensibilità e precisione, è in grado di fornire una misurazione puntuale del movimento del veicolo, in modo da rilevare urti di differenti entità. Allo stesso modo i frame generati dalla video camera sono di elevata risoluzione ed il loro salvataggio necessita di un quantitativo minimo di memoria, per merito del formato di codifica H.264. Come diretta conseguenza, lo spazio su disco occupato dai video è nettamente vantaggioso e permette al sistema di registrazione un numero elevato di secondi precedenti e successivi alla collisione. Inoltre, l'ampio campo visivo consente il monitoraggio completo dell'esterno del veicolo.

La scelta del Raspberry si è rivelata essere adeguata ai fini della progettazione e di una prima implementazione del sistema per merito della sua versatilità e semplicità di utilizzo. Nonostante ciò, in una reale implementazione, potrebbe non essere il dispositivo adatto per l'installazione a bordo del veicolo. Una migliore scelta sarebbe un single-board computer special-purpose con caratteristiche hardware in grado di fornire una maggiore potenza computazionale.

Infine, l'applicazione Android rappresenta l'unico punto di accesso ai dati memorizzati sul Raspberry ed offre all'utente la possibilità di visualizzare e scaricare le informazioni relative alle collisione. Il grafico delle accelerazioni ed il video sono in grado di fornire sia l'entità del danno ricevuto che una prova del veicolo responsabile, aumentando le probabilità di ottenere un risarcimento dalla compagnia assicurativa del colpevole. Le tecnologie di connettività a breve distanza impiegate aumentano la sicurezza del sistema e consentono all'utente di avere il pieno controllo dei propri dati.

Nella prossima sezione sono descritti i limiti riscontrati durante la realizzazione del sistema ed alcuni possibili sviluppi futuri con lo scopo di migliorarne l'implementazione.

4.1 Limiti e sviluppi futuri

Questo lavoro di tesi è nato con lo scopo di progettare un sistema a supporto dei proprietari dei veicoli che, in seguito ad una sosta, notano la proprio vettura danneggiata e non hanno alcun modo di ricostruire la dinamica dell'accaduto. Il sistema prevede l'installazione di un dispositivo a bordo del veicolo con lo scopo di monitorarne l'esterno durante l'assenza del conducente e registrare un video in seguito al rilevamento di una collisione.

Attualmente, l'utilizzo di un sistema del genere è vietato dalle vigenti norme sulla privacy in luogo pubblico a causa dell'impiego di una video camera che monitora l'esterno del veicolo. La diffusione del video potrebbe invadere la privacy altrui, tutela dati personali.

I possibili sviluppi futuri riguardano in particolar modo l'installazione fisica del dispositivo a bordo del veicolo. In questo lavoro, infatti, il sistema è stato utilizzato e testato in maniera del tutto indipendente dalla vettura. La prima considerazione è relativa alle condizioni di avvio e arresto del sistema, che possono essere collegate all'attività dei veicolo. L'utilità del sistema è rilevante solo in assenza del conducente e lo spegnimento del veicolo è la condizione ideale per avviarlo. La fine dell'attività del sistema può avvenire in seguito all'accensione della vettura, ma potrebbe essere anche temporizzata per evitare un consumo eccessivo della batteria.

Con la possibilità di installare fisicamente il dispositivo a bordo del veicolo, è possibile migliorare il rilevamento dei falsi positivi, che possono provocare la registrazione di un video anche in assenza di collisione: nell'ipotesi di avviare il sistema dopo lo spegnimento del veicolo, può capitare che l'accelerometro rilevi come urto anche una brusca chiusura delle portiere. Una semplice soluzione prevede l'inserimento di un ritardo nell'avvio del sistema, in modo che questo genere di eventi non venga rilevato come una collisione. In alternativa, si può utilizzare l'applicazione per valutare la vicinanza dell'utente al veicolo, ad esempio tramite l'utilizzo del Bluetooth: il dispositivo a bordo della vettura può iniziare la fase di attività solo se non rileva lo smartphone dell'utente nelle vicinanze.

Bibliografia

[1] A sudden bang when parking https://www.allianz.com/en/press/news/commitment/community/150505-a-sudden-bang-when-parking.html

[2] Descrizione del Raspberry https://it.wikipedia.org/wiki/Raspberry_Pi

[3] Raspberry PI 3 B+ hardware https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

[4] Android Operating System https://it.wikipedia.org/wiki/Android

[5] Open Handset Alliance https://it.wikipedia.org/wiki/Open_Handset_Alliance

[6] Qt Toolkit https://en.wikipedia.org/wiki/Qt_(software)

[7] JavaScript Object Notation https://www.json.org/

[8] A Java serialization/deserialization library to convert Java Objects into JSON https://github.com/google/gson

[9] I^2C bus https://it.wikipedia.org/wiki/I%C2%B2C

[10] Protocollo I^2C http://www.crivellaro.net/wp-content/uploads/2017/03/Protocollo-di-comunicazione-I2C.pdf

[11] Bit di Start e Stop http://www.crivellaro.net/wp-content/uploads/2017/03/Protocollo-di-comunicazione-I2C.pdf#%5B%7B%22num%22%3A34%2C%22gen%22%3A0%7D%2C%7B%22name%22%3A%22XYZ%22%7D%2C54%2C278%2C0%5D

[12] Bit di Acknowledge http://www.crivellaro.net/wp-content/uploads/2017/03/Protocollo-di-comunicazione-I2C.pdf#%5B%7B%22num%22%3A36%2C%22gen%22%3A0%7D%2C%7B%22name%22%3A%22XYZ%22%7D%2C54%2C405%2C0%5D

[13] Scrittura di un registro http://www.crivellaro.net/wp-content/uploads/2017/03/Protocollo-di-comunicazione-I2C.pdf#%5B%7B%22num%22%3A39%2C%22gen%22%3A0%7D%2C%7B%22name%22%3A%22XYZ%22%7D%2C54%2C401%2C0%5D

[14] Lettura di un registro http://www.crivellaro.net/wp-content/uploads/2017/03/Protocollo-di-comunicazione-I2C.pdf#%5B%7B%22num%22%3A39%2C%22gen%22%3A0%7D%2C%7B%22name%22%3A%22XYZ%22%7D%2C54%2C297%2C0%5D

[15] Segnale di Repeated Start http://www.crivellaro.net/wp-content/uploads/2017/03/Protocollo-di-comunicazione-I2C.pdf#%5B%7B%22num%22%3A42%2C%22gen%22%3A0%7D%2C%7B%22name%22%3A%22XYZ%22%7D%2C54%2C771%2C0%5D

[16] Specifiche della videocamera http://www.webcamerausb.com/elpusbfhd04hl36-p-89.html

[17] Tipi di fotogramma nella compressione video https://it.wikipedia.org/wiki/Tipi_di_fotogramma_nella_compressione_video
[18] Fritzing Software Tool http://fritzing.org/home/
[19] Mappa dei registri del sensore MPU https://www.invensense.com/wp-content/uploads/2015/02/MPU-6500-Register-Map2.pdf
[20] Configurazione del sensore MPU https://gzuliani.bitbucket.io/arduino/arduino-mpu6050.html
[21] MPU datasheet https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf
[22] Condition variable https://it.cppreference.com/w/cpp/thread/condition_variable
[23] FFmpeg https://ffmpeg.org/
[24] Advertising Data Type https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile
[25] Distorsione ottica https://en.wikipedia.org/wiki/Distortion_(optics)
[26] Brown, Duane C. (May 1966). "Decentering distortion of lenses" (PDF). Photogrammetric Engineering. 32 (3): 444–462. https://web.archive.org/web/20180312205006/https://www.asprs.org/wp-content/uploads/pers/1966journal/may/1966_may_444-462.pdf
[27] de Villiers, J. P.; Leuschner, F.W.; Geldenhuys, R. (17–19 November 2008). "Centi-pixel accurate real-time inverse distortion correction" (PDF). 2008 International Symposium on Optomechatronic Technologies. http://researchspace.csir.co.za/dspace/bitstream/handle/10204/3168/De%20Villiers_2008.pdf
[28] Fitzgibbon, A. W. (2001). "Simultaneous linear estimation of multiple view geometry and lens distortion". Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). IEEE. https://ieeexplore.ieee.org/document/990465/references
[29] Bukhari, F.; Dailey, M. N. (2013). "Automatic Radial Distortion Estimation from a Single Image" (PDF). Journal of mathematical imaging and vision. Springer. http://www.cs.ait.ac.th/~mdailey/papers/Bukhari-RadialDistortion.pdf
[30] Correzione di un'immagine fisheye https://www.shadertoy.com/view/4sXcDN