POLITECNICO DI TORINO

Corso di laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

# Sistema di guida autonoma basato su Fog Computing


**Relatore**
prof. Fulvio Risso

**Candidato**
Andrea Alfò

Ottobre 2017

Alla mia famiglia.

# Indice

# Elenco delle figure

# Capitolo 1

# Introduzione

Una nuova architettura si pone come strato intermedio tra il Cloud e l'*Internet of Things*. Il suo nome è Fog Computing ed ha come scopo quello di mettere a disposizione risorse, tipiche del Cloud, proprio dove servono. L'utilizzo di questa nuova tecnologia permette di ottenere tutta una serie di vantaggi:

- Riduzione della latenza: le azioni vengono intraprese in prossimità del luogo d'azione.
- Risparmio di banda: le informazioni vengono aggregate e spedite solo quando necessario.
- Maggior scalabilità: è possibile aggregare più applicazioni sotto un unico nodo o una confederazione di nodi, il tutto senza congestionare troppo la rete.

Nell'era dell'*Industry 4.0*, sono molti gli scenari in cui il Fog Computing trova la sua applicazione e, grazie alla bassa latenza, è in grado di funzionare in contesti real-time, come ad esempio la **guida autonoma**.

L'idea di auto che *si guidano da sole* è presente da tempo nell'immaginario collettivo: c'è chi la ritiene il futuro della mobilità, chi si oppone fermamente e chi crede semplicemente che ci siano troppi problemi tecnologici ed etici per la sua realizzazione. In realtà esistono già diversi prototipi di veicoli *self-driving* e nelle ultime automobili lanciate sul mercato la presenza di sistemi intelligenti, quali park assist o cruise control, è diventata molto massiccia.

Nel campo dell'agricoltura sono state introdotte numerose innovazioni tecnologiche, spingendo la produttività a dei livelli sempre più alti. La guida autonoma potrebbe essere il prossimo passo per raggiungere nuovi traguardi e la ricerca in questo campo è ancora agli albori.

Questa tesi ha come obiettivo la realizzazione di un piccolo prototipo di trattore con guida autonoma, sfruttando l'architettura del Fog Computing e mostrando i vantaggi ad essa legati. Il Cloud diventa anch'esso parte integrante della soluzione,

ma ha come unico scopo quello di dare un valore aggiunto al sistema, che rimane tuttavia indipendente ed autonomo.

Questo elaborato è strutturato come segue:

- **Capitolo 2**: descrive in maniera esauriente il concetto del Fog Computing, mostrando vari scenari di utilizzo e focalizzando l'attenzione sui vantaggi che ne derivano.
- **Capitolo 3**: mostra in dettaglio la tecnologia offerta da Nebbiolo Technologies ed esamina la configurazione del fogNode utilizzato per la realizzazione del prototipo.
- **Capitolo 4**: fornisce la classificazione dei veicoli autonomi ed analizza le tecnologie presenti sul mercato, distinguendo quelle per l'uso stradale da quelle agricole.
- **Capitolo 5**: descrive i diversi applicativi utilizzati per la realizzazione del prototipo, mostrando dettagliatamente le loro funzionalità e le loro caratteristiche.
- **Capitolo 6**: spiega tutti i componenti realizzati ai fini della soluzione, si occupa di chiarire quali sono i loro compiti e come vengono svolti, sottolineando le motivazioni che hanno spinto a determinate scelte implementative a discapito di altre.
- **Capitolo 7**: fornisce una panoramica dei risultati ricavati misurando le performance del sistema, soprattutto in termini di latenza e velocità di esecuzione.
- **Capitolo 8**: espone le conclusioni e presenta suggerimenti per le future versioni di questa soluzione, mostrando anche le difficoltà legate alla tecnologia adottata.

# Capitolo 2

# Fog Computing

In questo capitolo viene spiegato il concetto del Fog computing, quali sono le differenze e i vantaggi rispetto al Cloud tradizionale e vengono esaminati in dettaglio alcuni use cases significativi.

## 2.1 Definizione

Il Fog computing (o Edge computing) è un'*architettura orizzontale*, a livello di sistema, utile a distribuire risorse e servizi di calcolo, immagazzinamento di dati, controllo e funzionalità di rete più vicino agli utilizzatori (dispositivi), lungo l'infrastruttura che li connette al Cloud [1].

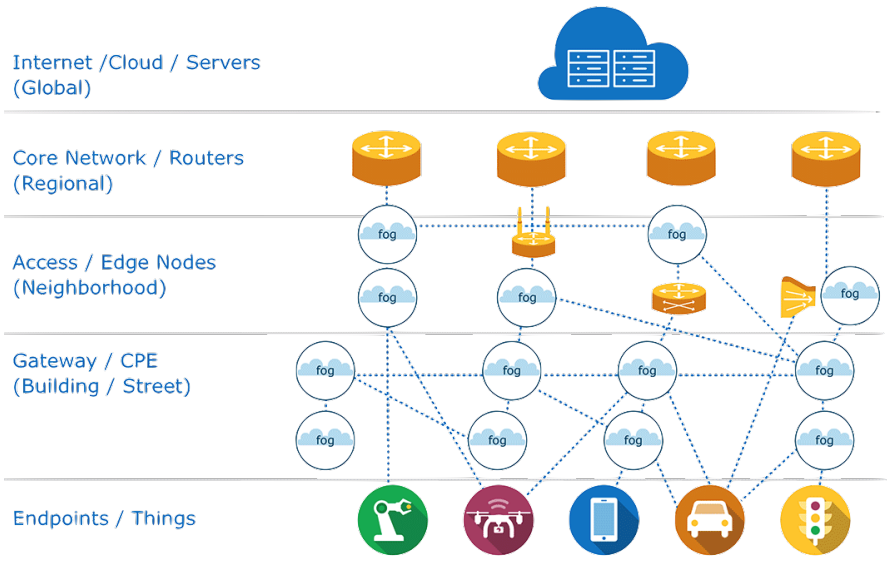



Figura 2.1. Architettura Fog Computing (Fonte: OpenFog Consortium [1]).

Il Fog computing, dunque, estende il modello tradizionale del Cloud avvicinando in parte risorse e servizi all'edge della rete: è metaforicamente *una "nuvola" più vicina al suolo*. Un esempio dell'architettura è mostrato in Figura 2.1.

Fog e Cloud basano il loro funzionamento sulla stessa tipologia di risorse (capacità di elaborazione e immagazzinamento dati, funzionalità di rete) e condividono gli stessi meccanismi di funzionamento (virtualizzazione, multi-tenancy), tuttavia esiste una fondamentale differenza che dà al Fog la sua ragion d'essere.

Il Cloud è un'architettura fortemente centralizzata dove grosse moli di dati, provenienti dai dispositivi che ne utilizzano i servizi, sono costretti a transitare lungo il core della rete fino a raggiungere i data-center di un service provider. Contrariamente, il Fog offre risorse e servizi seguendo la naturale distribuzione geografica degli utilizzatori. Nonostante questa apparente controtendenza presente nei due paradigmi, Fog e Cloud traggono reciproci benefici dalla loro mutua interazione.

Le applicationi real-time o quelle installate su dispositivi con mobilità veloce o scarsa connettività non possono fare affidamento sul Cloud, ed è qui che il Fog computing entra in gioco: la vicinanza all'edge implica ritardi molto bassi; l'elaborazione locale consente l'invio di una quantità ridotta di dati ai datacenter, con relativo risparmio di banda e costi; la perdita temporanea di connettivà non pregiudica lo svolgimento delle operazioni in corso. Tutti questi vantaggi vengono riassunti, dall'OpenFog Consortium, con il termine ***SCALE*** [2]:

- **S**ecurity: dati sensibili elaborati localmente su un nodo fidato
- **C**ognition: approccio client-centric che favorisce una maggiore autonomia dal Cloud
- **A**gility: innovazione rapida e applicazioni scalabili utilizzando un'infrastruttura comune
- **L**atency: basso ritardo di comunicazione
- **E**fficiency: condivisione di risorse localmente inutilizzate, derivanti dalla partecipazione dei dispositivi finali

## 2.2 Use cases

In questa sezione verranno descritti degli use cases [3] rilevanti per mettere in luce i vantaggi e le caratteristiche chiave del Fog computing.

### 2.2.1 Use case 1: Sistema per il controllo intelligente del traffico

La congestione del traffico è un problema rilevante nei grandi agglomerati urbani e può portare a paralizzare le grandi città. Gli impatti negativi che una situazione di congestione stradale ha sulla comunità sono diversi: non solo ritardi e disguidi, ma anche stress sui veicoli e sui conducenti, ostacolamento di mezzi impegnati in emergenze, maggior consumo di carburante e conseguente aumento delle emissioni. Un sistema per il controllo intelligente del traffico si occupa della prevenzione degli incidenti, di mantenere costante il flusso del traffico e di raccogliere dati rilevanti per migliorare il sistema stesso. Questi tre obiettivi sono molto diversi tra loro dal punto di vista della scala del tempo: mentre il primo richiede tempi di reazione tipici di un sistema real-time; il secondo può tollerare ritardi decisamente maggiori; infine l'ultimo riguarda l'analisi a lungo termine dei dati raccolti e delle decisioni prese dal sistema. L'architettura flessibile e geograficamente distribuita del Fog Computing permette di soddisfare questi requisiti favorendo tempi di latenza molto bassi, integrazione e analisi di dati provenienti da dispositivi eterogenei e precedentemente non interconnessi (dispositivi di controllo del traffico e segnaletica elettronica, sensori sul ciglio della strada e dispositivi a bordo dei veicoli) e demandando al Cloud l'analisi in batch delle performance del sistema.

### 2.2.2 Use case 2: Consegna dei pacchi tramite droni

Ogni giorno miliardi di pacchi vengono distribuiti in tutto il mondo e i costi legati al loro trasporto (tramite treni, aerei, furgoni, ecc...) diventano sempre maggiori. La tecnologia IoT ha migliorato alcuni aspetti di questo settore, introducendo una migliore gestione del traffico e ottimizzando i percorsi da seguire. L'idea di utilizzare dei droni UAV per la consegna dei pacchi diventa sempre più concreta per abbattere ulteriormente i costi e migliorare il servizio, garantendo tempi di consegna inferiori anche ai 30 minuti. Le sfide da affrontare sono molteplici:

1. Collisioni e sicurezza: in uno scenario in cui milioni di droni volano contemporaneamente, il pericolo di una collisione tra loro diventa concreto; in aggiunta piccoli aerei, uccelli o grandi edifici rappresentano fin da subito una minaccia reale. Diventa quindi fondamentale riuscire ad avere tempi di reazione dell'ordine di pochi millisecondi per evitare incidenti.

2. Banda limitata: la costante comunicazione di dati e il monitoraggio in tempo reale dei droni potrebbero saturare la banda che si ha a disposizione. Nelle zone in cui la connettività è limitata o assente bisognerebbe utilizzare canali satellitari per comunicare con il Cloud, il che comporterebbe costi proibitivi.

3. Hub per il controllo: così come per gli aerei di linea e quelli commerciali, anche i droni hanno bisogno di un sistema di controllo per coordinare i voli dei vari soggetti.

Supponiamo di voler utilizzare il Cloud: se si considera che un drone può volare a 160km/h e che il minor RTT ottenibile è di circa 80ms, significa che tra una comunicazione e l'altra il drone ha percorso la distanza di circa 214m [4]. Far volare il drone per 214m (nel caso migliore) senza alcun tipo di controllo è veramente impensabile, questo ci fa capire come la scelta del Cloud in questa circostanza non può essere la soluzione. Utilizzando il Fog computing nella comunicazione tra il drone e il "centro di controllo", i comandi di update possono essere inviati con un ritardo così basso da far muovere il drone di pochi centimetri prima che il prossimo messaggio gli venga consegnato. Il Cloud rimane tuttavia sempre presente e i dati di monitoraggio vengono inviati, in questo caso senza fretta, ai grossi datacenter per essere poi elaborati e/o salvati.

### 2.2.3 Use case 3: Videosorveglianza

Ai fini di garantire sicurezza alle persone, alle cose e ai luoghi negli ultimi anni sono state installate sempre più telecamere di sorveglianza. Una singola telecamera può generare (dipende dalla risoluzione e dalla qualità scelta) anche 1TB di dati al giorno. Le telecamere dei sistemi di sorveglianza generano dati che devono essere analizzati in tempo reale per garantire la pubblica sicurezza. Il numero crescente di telecamere e l'aumentare della risoluzione e della qualità dei video prodotti, hanno generato un enorme flusso di dati verso il Cloud. I problemi quindi legati a questo scenario sono molteplici:

1. Alta definizione: il modello tradizionale del cloud era stato realizzato per sistemi a bassa risoluzione, attualmente non è difficile trovare in commercio telecamere a basso costo che generano filmati ad una risoluzione Full-HD (1080p) o 4K, questo aumenta drasticamente la dimensione dei dati da trasmettere.

2. Tempi di risposta: nel caso di un'intrusione all'interno di un edificio, è necessario accedere immediatamente ai dati per poter permettere alla sicurezza di localizzare la minaccia in tempi brevi, lo stesso discorso vale per un fuggitivo a bordo di un mezzo che scappa per le vie di una città.

3. Scalabilità: nel caso di luoghi con numerosi dispositivi di acquisizione video (es: aeroporti, stazioni, stadi) la quantità di dati da trasmettere al cloud supera di gran lunga la banda che si ha a disposizione.

Il Fog computing forma una maglia di nodi che intelligentemente partizionano il processamento video tra i nodi fog stessi e il cloud, garantendo il tracciamento real-time,

il rilevamento di anomalie e la raccolta di informazioni significative. Gli algoritmi di analisi video possono essere installati direttamente sui nodi fog, riducendo il traffico verso il cloud e garantendo tempi di latenza ridotti.

## 2.3 Fog Computing e guida autonoma

L'architettura del Fog Computing è molto versatile e il suo campo di applicazione è veramente vasto. In particolare c'è un nuovo settore, quello della guida autonoma, che si addice proprio a questa nuova tecnologia. La bassa latenza, l'elevata scalabilità, la sicurezza e le altre proprietà del Fog rispecchiano le naturali necessità della guida autonoma.

Lo scenario tipico è quello di un veicolo che deve essere guidato all'interno di un percorso ed evitare gli ostacoli che incontra lungo il proprio cammino. In questa tesi è stato esaminato un particolare caso, quello di un trattore che deve seguire dei lavori in un campo, evitando di colpire oggetti o persone.

La sfida principale sta nel fatto che tutta l'intelligenza deve risiedere a bordo e deve riuscire a funzionare anche quando la connettività non è presente o è intermittente. I veicoli che girano su strada e che sono concepiti per essere *driverless*, sfruttano una continua cooperazione con le informazioni e gli ordini che arrivano dal Cloud. In questo scenario non sono presenti grosse reti cellulari nei paraggi ed in campagna, avere una connessione ad Internet, è qualcosa di raro.

L'idea fin da subito è stata quella di realizzare un sistema totalmente **indipendente** dal Cloud, ma che si appoggi ad esso qualora ce ne fosse la necessità. Nei prossimi capitoli questo tema verrà ripreso più volte e verranno mostrati tutti i componenti del sistema di guida autonoma che lo realizzano.

# Capitolo 3

# La tecnologia di Nebbiolo Tecnologies

La piattaforma di Fog Computing sviluppata da Nebbiolo Technologies è composta da piccoli server distribuiti all'edge della rete, con gestione delle risorse tipiche del Cloud e funzionlità real-time, al fine di colmare il gap tra i dispositivi IoT e l'infrastruttura Cloud [5].

## 3.1 Cenni sull'azienda

Nebbiolo Tecnologies, start-up californiana con radici italiane, nasce nel 2015 con l'obiettivo di progettare e costruire apparati per il Fog computing. L'idea di questa emergente tecnologia è nata nel lontano 2010 dalla mente dell'attuale CEO e founder Flavio Bonomi, quando ancora ricopriva il ruolo di vicepresidente del dipartimento di ricerca di Cisco Systems. Gli ingegneri di Nebbiolo Tecnologies sono pertanto i pionieri del Fog computing con la loro prima serie di dispositivi, denominati fogNode$^{TM}$, lanciati sul mercato il 9 febbraio 2017. Dispositivi che, essendo in grado di fornire bassa latenza di comunicazione e capacità di elaborazione real-time, hanno attirato l'attenzione di grandi player nel campo dell'automazione industriale come Kuka e TTTech.

## 3.2 Architettura

L'architettura della piattaforma di Fog computing di Nebbiolo Technologies è suddivisa, come mostrato in Figura 3.1, in tre livelli: i nodi hardware chiamati fogNode$^{TM}$, il livello software fogOS$^{TM}$ e il sistema di gestione fogSM$^{TM}$ (o NSM - Nebbiolo System Management).

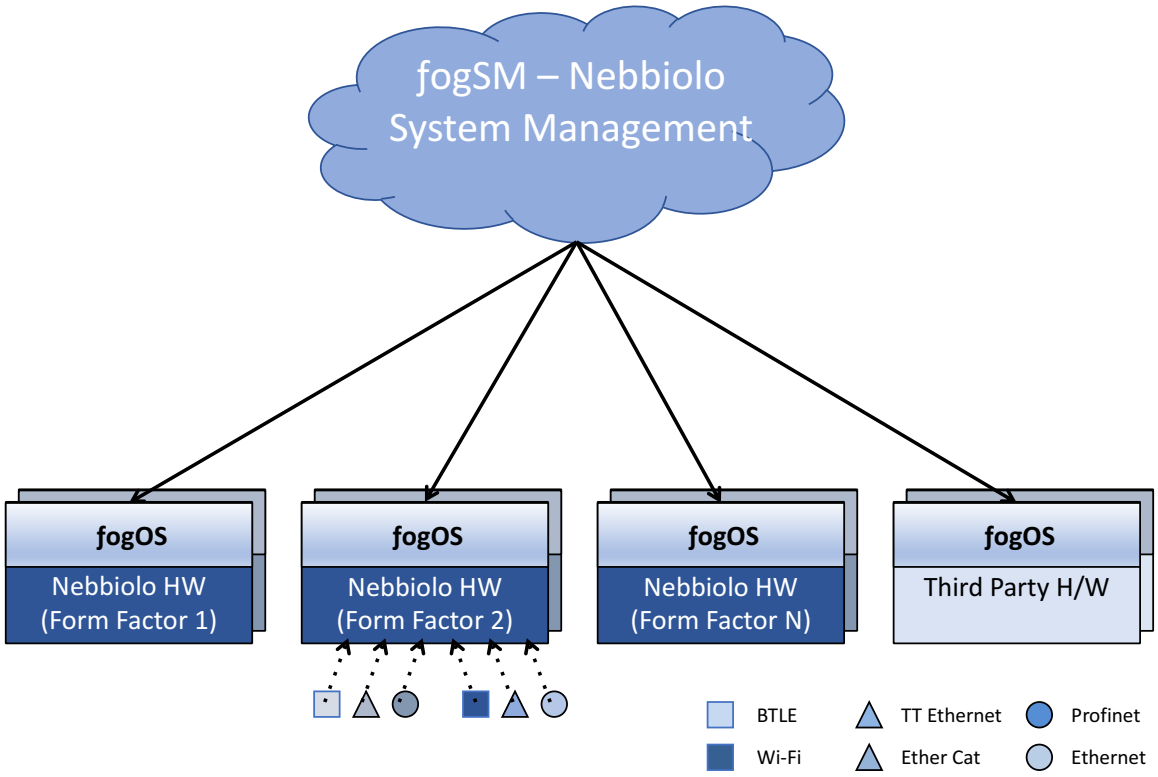



Figura 3.1. Architettura della piattaforma Fog di Nebbiolo Technologies.

### 3.2.1 Il fogNode

I fogNode$^{TM}$ si basano su un'architettura modulare e pertanto possono essere costruiti con diversi fattori di forma e diverse funzionalità (Figura 3.2) in base alle esigenze, permettendone il loro utilizzo in differenti ambiti applicativi. Ogni fogNode, dunque, può essere composto da uno o più fogLet: si tratta di un modulo con risorse computazionali, di storage e connettività indipendenti. I fogLet utilizzano CPU di elevate prestazioni e dispongono di storage veloci (SSD). Sono equipaggiati con switch Ethernet con capacità TSN - *Time-Sensitive Networking* - per fornire connettività con latenze deterministiche, fondamentali per un utilizzo real-time. Per espandere la connettività possono opzionalmente essere equipaggiati con moduli WiFi ed LTE [6].

La versione utilizzata in questa tesi è il Nebbiolo fogNode NFN 300 che comprende al suo interno un solo modulo NFL-1000-C, dotato di 6 porte Gigabit Ethernet IEEE 802.3ab, una porta VGA, una USB 2.0 ed una porta seriale RJ45. All'interno è presente un SSD SATA 3 da 120GB ed un processore Dual Core Intel Haswell di quarta generazione (Core-i5 4402E). Il consumo energetico è di circa 100W, pesa 7.6KG ed è certificato IP20 (apparecchio da interno) [7].

| fogNode™ Series | |
|---|---|
|  | • Fan cooled chassis<br>• Per Slot: 4-8 core x86 i5/i7,<br>• 128-512G Storage,<br>• 8-16G memory,<br>• LTE and WiFi<br>• Secure Hardware,<br>• Real Time capable with embedded Switch<br>• 3 slots connected backplane for High Availability, Scale and Aux cards (e.g. GPU, Storage, Safety) |
|  | • Fanless, 24V DC powered<br>• 4-8 core x86 Corei5/i7,<br>• 128-512G Storage,<br>• 8-16G memory,<br>• LTE and WiFi,<br>• Secure Hardware,<br>• Real Time capable with Embedded Switch |
|  | • 4 core Atom,<br>• 32-128G Storage,<br>• 8G memory,<br>• LTE and WiFi,<br>• Secure Hardware,<br>• Real Time capable with Embedded Switch |
|  | • 2 core Atom, 32G Storage, 4-8G memory,<br>• 3G and WiFi Gateway functions |

Figura 3.2. Diverse configurazioni di fogNode™ (Fonte: Nebbiolo.tech [6]).

### 3.2.2 fogOS

Il **fogOS** è il sistema operativo realizzato da Nebbiolo per i nodi Fog. E' composto da due parti (Figura 3.3): la prima, basata sulla distribuzione CentOS 7.3 e kernel Linux 3.10, racchiude al proprio interno tutto l'insieme di librerie e software utili

per l'esecuzione degli applicativi in locale; la seconda, invece, gira sul Cloud e si occupa della gestione globale dei nodi.

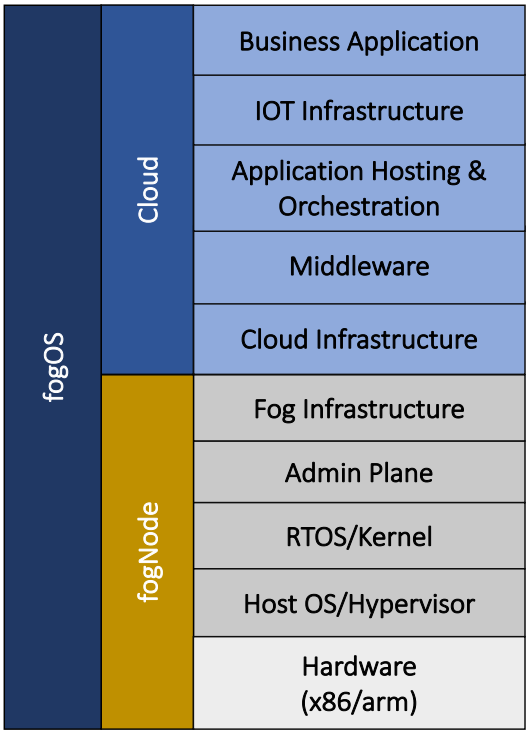



Figura 3.3. Lo stack che compone il fogOS.

- **Hypervisor**: un gradino sopra l'hardware è presente un hypervisor KVM. Al proprio interno custodice il sistema operativo base del nodo ed ha il compito di occuparsi della gestione del ciclo di vita di virtual machine e container.
- **RTOS**: sistema operativo base dotato di capacità di Real-Time e di un ricco set di funzionalità di virtualizzazione.
- **Admin Plane**: si occupa della gestione delle connessioni TLS o VPN, dei certificati e della registrazione del device nel sistema di management. Fornisce inoltre un meccanismo per la generazione di allarmi e per il monitoraggio dei guasti interni al nodo. E' in questo layer che avviene il riconoscimento di periferiche esterne, o *asset*, e la gestione della rete e dello storage.
- **Fog Infrastructure**: mette a disposizione un client OPC UA, basato sul protocollo TCP, che permette di realizzare un ponte tra i componenti di automazione integrati e i sistemi operativi tradizionali. Questo client fornisce

tutte le informazioni desiderate a quegli applicativi che dispongono di opportune autorizzazioni [8]. In questo livello avviene inoltre la gestione del TSDB (Time-Series Database)[1], dei sensori e del bus dati.

- **Cloud Infrastructure**: a partire da questo modulo si parla di componenti che vengono eseguiti totalmente sul Cloud. Si occupa della catalogazione dei nodi, degli upgrade software e dello scaling orizzontale. Esegue un server OPC UA al quale i nodi si collegano per ottenere informazioni e dati. Fornisce inoltre diverse API server.

- **Application Hosting & Orchestration**: mette a disposizione un *App Store* dal quale è possibile effettuare il provisioning delle applicazioni. Si occupa della gestione di certificati, policy, RBAC (controllo accessi), topologia nodi e analitycs.

### 3.2.3 fogSM

Il fogSM (o NSM - Nebbiolo System Management) fornisce un'interfaccia di gestione unificata per una federazione di fogNode. Si raggiunge attraverso un browser web e permette di eseguire diverse operazioni. E' formato da quattro tab:

1. **Dashboard**: subito dopo il login, si accede a questa pagina. Da qui è possibile visualizzare lo stato dei nodi e degli asset (attivo, inattivo, non configurato), controllare le ultime notifiche e visualizzare la disposizione geografica dei nodi all'interno di una mappa.

2. **Inventory**: vengono mostrati i nodi aggregati in base alle policy scelte. E' possibile espandere le varie voci per poter visualizzare quali container o applicazioni sono in esecuzione. Da questa schermata si può eseguire il deploy di un'applicazione, il suo undeploy, la cancellazione di un nodo o la configurazione dei parametri da passare. Sono disponibili inoltre dettagli aggiuntivi riguardanti la versione del sistema installato, la configurazione di rete e i vari log generati (Figura 3.4).

3. **Application Store**: permette di caricare applicazioni, macchine virtuali e container docker all'interno dell'NSM. E' possibile definire categorie di applicazioni ed autorizzare l'onboard solo per alcune di esse. I container possono essere caricati direttamente dal Docker Hub, specificando repository e tag. Tutti gli eventuali settaggi, compreso il port mapping, vengono gestiti all'interno di questo tab. Il provisioning avviene come mostrato in Figura 3.5: l'applicativo

[1]Database ottimizzato per la gestione di dati e array di numeri indicizzati in base al tempo.

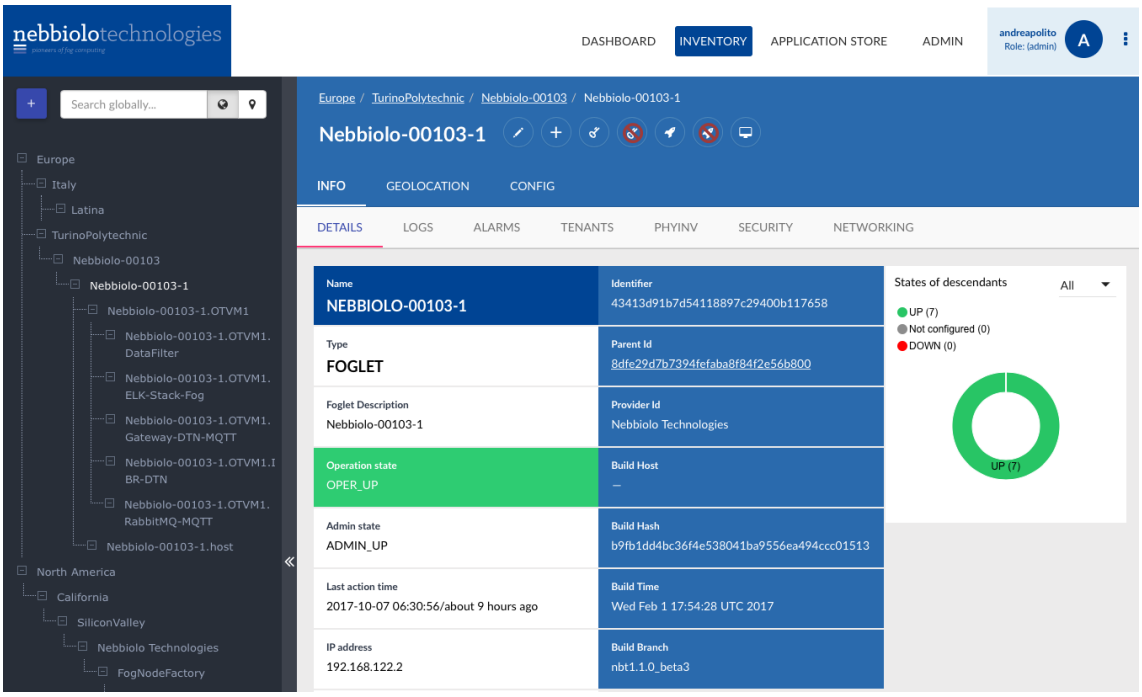



Figura 3.4. Screenshot dell'interfaccia web del fogSM - sezione "inventario".

deve essere prima caricato all'interno dell'Application Store; successivamente si sceglie l'asset o il nodo su cui far eseguire il deploy.

4. **Admin**: in questa ultima schermata è possibile configurare notifiche, ruoli, aggiungere o rimuovere utenti e controllare le infomazioni relative all'NSM stesso.

## 3.3 Configurazione del fogNode

In questa sezione viene descritta la configurazione del nodo utilizzato per lo sviluppo di questa tesi, collocato all'interno del laboratorio del NetGroup presso il Politecnico di Torino.

### 3.3.1 Virtual machine

Di base è presente una macchina virtuale, denominata ***AdminVM***, che rappresenta l'interfaccia per la gestione del fogLet da remoto o dall'apposita porta di management, collocata sullo switch dello stesso. La AdminVM contiene la versione 1.12.1

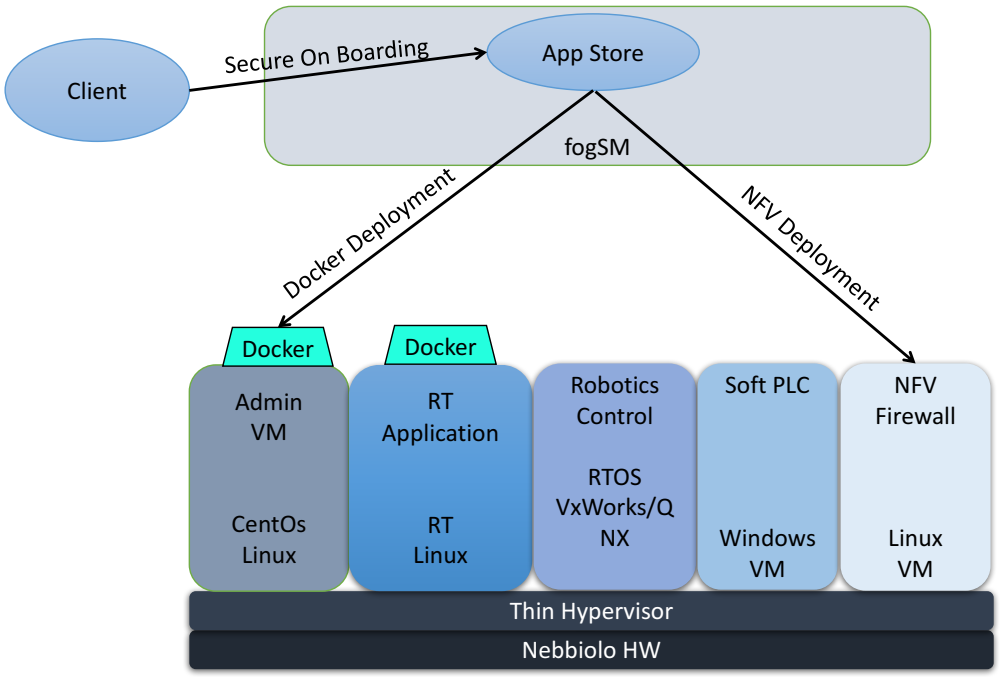



Figura 3.5. Esempio di provisioning di un'applicazione (Fonte: Nebbiolo Technologies).

di Docker e tutto lo stack software necessario per eseguire il deployment di applicazioni dal fogSM, nonché il message broker RabbitMQ. Questa macchina virtuale deve essere sempre accessibile da parte di Nebbiolo per poter aggiornare il software del sistema e per poter sincronizzare il nodo con il Nebbiolo System Manager. Per questioni di privacy è quindi sconsigliato effettuare il deployment delle proprie applicazioni all'interno di essa.

A tale scopo è preferibile istanziare altre virtual machine, denominate ***OTVM***, preposte a contenere applicazioni e servizi. L'immagine di queste OTVM è fornita da Nebbiolo Technologies nell'Application Store e contiene anch'essa Docker e tutto il software necessario per eseguire il deployment di applicazioni da remoto. A differenza della AdminVM, nell'immagine base della OTVM non è compreso RabbitMQ, ma questo può essere facilmente istanziato dall'NSM sottoforma di container Docker. Nel caso specifico del nodo utilizzato è stata istanziata una singola OTVM.

Generalmente è presente anche un'altra macchina virtuale, la *Forwarder Virtual Machine* – ***FWD VM***, che fornisce la funzionalità di inoltro del traffico verso Internet a tutte le OTVM. Quest'ultime, infatti, non sono connesse verso l'esterno, ma possono comunicare tra loro all'interno del fogLet che le ospita. Questa scelta progettuale nasce dalla volontà di isolare le OTVM, sulle quali solitamente vengono

ospitate applicazioni e servizi che elaborano dati sensibili. Eventualmente solo le informazioni che devono uscire dal nodo saranno inoltrate, tramite il meccanismo del message brokering, alla FWD VM. Anch'essa è dotata, come la AdminVM, del message broker RabbitMQ. Grazie agli svariati protocolli supportati e alla flessibilità dell'architettura publish/subscribe, esso rappresenta il punto d'uscita verso svariati servizi sul Cloud. Per le future release software sono previsti appositi *"Connector"* per facilitare il collegamento alle piattaforme Cloud dei maggiori service providers (Microsoft Azure, Amazon Web Services, Google Container Engine).

### 3.3.2 Configurazione di rete

Il fogLet presenta 6 interfacce Gigabit Ethernet, 4 sono collegate ad uno switch interno, le restanti due sono dirette. Le porte dello switch prendono i nomi S0, S1, S2 ed S3, mentre quelle dirette E0 ed E1. La connettività ad Internet deve essere fornita attraverso la porta E0, mentre la E1 viene utilizzata per il management locale.

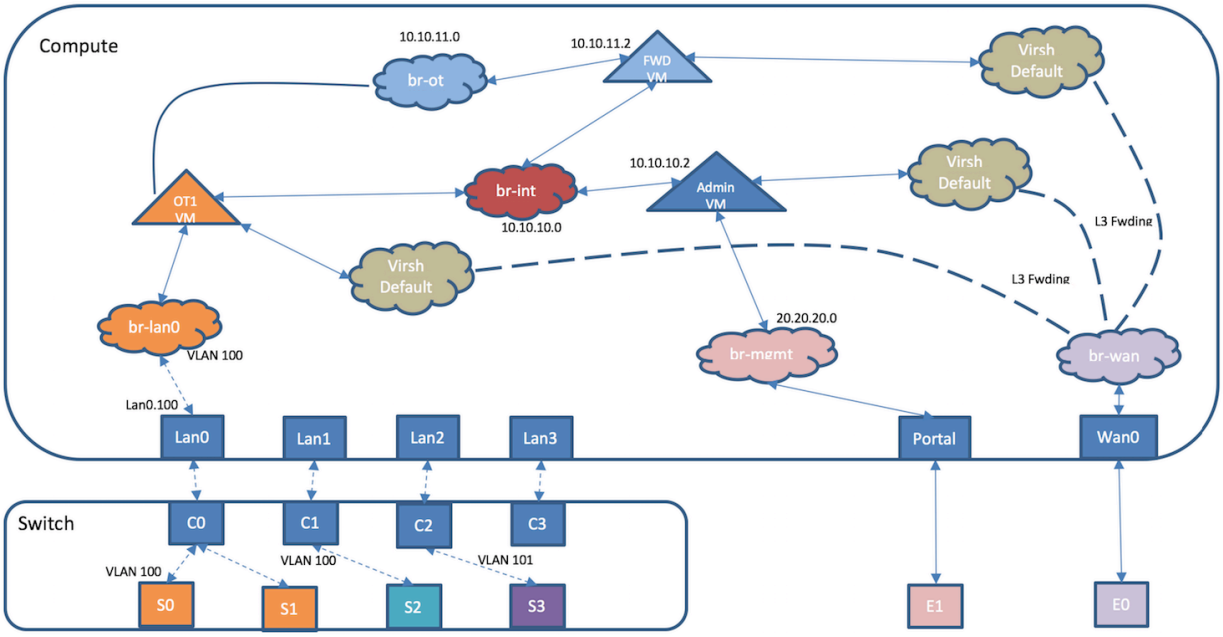



Figura 3.6. Architettura di rete e configurazione interna del fogLet (Fonte: Nebbiolo Technologies)

In Figura 3.6 viene mostrata la specifica configurazione di rete del fogLet a disposizione, da cui si può notare la presenza di numerose "nuvole" che interconnettono

le varie macchine virtuali: si tratta di bridge Open vSwitch (OVS). Di seguito si elencano i bridge presenti e le loro rispettive funzioni:

- "**br-int**" usato per la comunicazione interna tra le VM;
- "**br-mgmt**" connette la AdminVM alla porta fisica E1 per le funzionalità di gestione;
- "**br-wan**" fornisce la connettività verso Internet;
- "**br-lan0**" utilizzato per connettere le porte fisiche S0 ed S1 dello switch con la OTVM1;
- "**br-ot**" collega tutte le OTVM (in questo caso solo una) con la Forwarder VM (FWD VM);

Sempre in Figura 3.6 si può notare come l'unica OTVM presente, diversamente da quanto descritto nella sottosezione 3.3.1, sia collegata al bridge denominato "br-wan". Nella nostra configurazione, infatti, la OTVM1 può fare uscire traffico verso Internet senza passare dalla FWD VM. Si tratta, però, di una "feature" di testing e dovrebbe essere eliminata nelle future release software.

Si noti, inoltre, come le porte dello switch S2 ed S3 siano al momento prive di alcun collegamento con gli elementi interni del fogLet. Le porte S0 ed S1 dello switch, come detto, sono invece connesse alla OTVM1 attraverso il bridge "br-lan0". Nello specifico le porte S0 ed S1 sono rispettivamente collegate alle interfacce "eth0" ed "eth1" della OTVM1. Alla porta S0 è stato connesso un Access Point Wi-Fi, mentre sull'interfaccia eth0 è stato configurato un server DHCP: tutti i dispositivi che si connettono all'access point vengono configurati con un indirizzo appartenente alla rete privata 192.168.1.0/24, con l'ultimo ottetto nel range 100-199. L'interfaccia eth0 e l'access point hanno invece indirizzo rispettivamente 192.168.1.1 e 192.168.1.2.

# Capitolo 4

# Guida autonoma

Quello della guida autonoma è un tema che attira sempre più attenzioni. Aziende automobilistiche e dell'ambito IT, negli ultimi anni, hanno investito molte energie e risorse per la realizzazione di prototipi e sul mercato cominciano già a vedersi i primi risultati. D'altro canto l'Unione Europa, gli USA e molti stati in fase di crescita guardano con un certo interesse quale impatto possa avere la guida autonoma sulla sicurezza nelle strade, sui viaggi e sullo sviluppo urbano.

## 4.1 Classificazione dei veicoli autonomi

Oggigiorno sono presenti numerosi veicoli definiti *autonomi* o *automatici*, ma presentano notevoli differenze in base alla tecnologia utilizzata e all'intervento dell'uomo durante le varie operazioni. Per porre fine a tutta una serie di ambiguità, la *Society of Automotive Engineers*, **SAE**, ha proposto sei livelli di classificazione [9], partendo dei veicoli senza automazione fino a quelli totalmente guidati da un'intelligenza artificiale.

- **Livello 0**: nessun tipo di automazione. Rientrano in questa categoria tutti i veicoli in cui è il guidatore a monitorare la situazione e ad eseguire tutte le manovre durante il viaggio. Tuttavia in questa classe sono presenti anche i veicoli che dispongono di "aiuti" elettronici quali: sensori di parcheggio, sistema di anti bloccaggio dei freni (ABS), avviso di superamento della linea della carreggiata, sistemi anti slittamento (ESP) e sistema di controllo della stabilità (ESC).
- **Livello 1**: assistenza al guidatore. L'essere umano è responsabile di tutti gli aspetti della guida quali evitare gli ostacoli, attivare o disattivare i sistemi elettronici e condurre il veicolo secondo il percorso desiderato. Esistono però a bordo dei sistemi quali cruise control o park assist che possono intervenire nella guida solo in particolari circostanze.

- **Livello 2**: automazione parziale. I sistemi presenti in questo livello sono in grado sia di accelerare/frenare sia di cambiare la direzione delle ruote. Il guidatore ha la responsabilità di monitorare il veicolo e i sistemi di guida, e deve essere pronto ad agire in caso di emergenza. A questo stadio il guidatore può anche togliere le mani dal volante in numerose circostanze. Tuttavia bisogna essere sempre pronti ad intervenire. Alcuni esempi sono: sistema avanzato di parcheggio e il sistema per la guida assistita in caso di ingorgo stradale.

- **Livello 3**: automazione condizionata. I sistemi sono in grado di eseguire tutti gli aspetti della guida e di monitorare l'ambiente circostante sotto alcune condizioni. Il guidatore non ha la necessità di controllare sempre quello che sta accadendo, ma deve rimanere tuttavia pronto per agire. Il sistema avvisa preventivamente l'essere umano nel caso in cui sia richiesta la necessità di passare al controllo "manuale". In questo livello rientrano il *Traffic Jam Chauffeur*[1] e l'*Highway Chauffeur*[2].

- **Livello 4**: automazione elevata. Come nel livello 3, i sistemi di guida si occupano di tutto, ma qui il guidatore non è quasi mai necessario. Si disattivano solo in determinate circostanze o quando il pilota ha ripreso il controllo manuale del veicolo.

- **Livello 5**: automazione totale. Il guidatore non è più necessario in qualunque condizione di guida. Rientrano in questa categoria i cosiddetti veicoli "*self-driving*".

## 4.2 Stato dell'arte

Alcuni dei sistemi di guida autonoma elencati nei vari punti precedenti sono già presenti nel mercato, mentre altri sono ancora in fase di sviluppo. I sistemi di guida totalmente autonoma (livello 5) potranno essere disponibili tra pochi anni: le auto senza pilota, o *self-driving*, di Google girano già sulle strade della California, rispettando il codice della strada, controllando le luci dei semafori, evitando le altre auto e negoziando la precedenza negli incroci più complessi.

Esistono fondamentalmente due tipi di approcci verso la guida autonoma:

[1] Il guidatore, in condizioni di traffico, può delegare temporaneamente il controllo del veicolo a questo sistema. Si occupa di regolare automaticamente la velocità in base ai veicoli che precedono e ai limiti di velocità.

[2] A differenza del *Traffic Jam Chauffeur* è adatto a tutti i tipi di traffico. Mantiene la carreggiata anche in autostrada e può decidere di superare la linea per effettuare sorpassi, mettersi in una corsia più lenta o selezionare l'uscita di un'autostrada.

- **evoluzionario**: è tipico delle case automobilistiche. Consiste nel realizzare delle tecnologie che aumentino la sicurezza e riducano lo stress della guida. Procedendo quindi a piccoli passi fino a raggiungere, come ultimo step, la guida totalmente indipendente dall'essere umano.
- **rivoluzionario**: Google, in primis, ha dichiarato la volontà di realizzare un sistema di guida autonoma completo, rivoluzionando totalmente la concezione di auto e guidatore.

Sebbene i concetti di *veicolo autonomo* e *veicolo connesso* siano differenti, sono fortemente legati l'uno all'altro. Le tecnologie che connettono veicoli con altri veicoli o con infrastrutture, sono già in uso nelle auto "comuni" e rappresentano un elemento cruciale per lo sviluppo dell'automazione completa.

### 4.2.1 Sicurezza

La sicurezza sulla strada, la riduzione delle emissioni inquinanti e delle congestioni, sono i maggiori benefici che ci si aspetta dalla diffusione dei veicoli autonomi. Tuttavia bisogna analizzare e studiare a fondo i vari scenari. Per esempio, quello della **sicurezza** è un aspetto cruciale per la diffusione dei veicoli senza pilota. Sebbene si stimi che il 90% degli incidenti sia causato da errore umano, non è detto che si abbia una riduzione della stessa percentuale. Inoltre bisogna considerare anche i molti fattori etici legati alle decisioni che gli algoritmi installati sui veicoli dovranno prendere. Il sistema di guida autonoma potrebbe trovarsi in situazioni in cui l'incidente è inevitabile e non è concepibile sviluppare un software che decida chi sacrificare scegliendo in base all'età, al genere o al numero.

La diffusione dei veicoli semi o totalmente autonomi (livelli 4 e 5) non può avvenire istantaneamente. Oggigiorno, sulle strade di tutto il mondo, sono presenti veicoli moderni e molto sicuri che circolano insieme a veicoli ormai obsoleti, che non presentano nemmeno i sistemi di sicurezza base come l'ABS o gli Airbag. Lo stesso avverrà per questa nuova generazione: si stima che la coesistenza tra i veicoli autonomi e quelli tradizionali possa durare anche 30-40 anni; durante i quali gli errori umani continueranno ad esserci. Questo può comportare un aumento delle situazioni critiche, ad esempio le manovre improvvise compiute da un essere umano potrebbero non essere predette dal software di guida, causando sinistri gravi. Di conseguenza non è detto che l'introduzione sulle strade di sistemi autonomi faccia diminuire il numero di incidenti mortali, è necessario dunque studiare il fenomeno su larga scala prima di poter dare delle valutazioni corrette.

### 4.2.2 Regolamentazione

Un altro problema è quello legato alla **regolamentazione**. Bisognerà realizzare dei severi sistemi di validazione per permettere ad un veicolo autonomo di guidare

per le strade pubbliche. Dovranno essere standardizzati i test e le procedure da adottare. Questi dovranno essere ben chiari fin dalle fasi dello sviluppo per non veder fallire delle aziende che hanno investito tanto sulla creazione di un prodotto che verrà bloccato prima dell'immissione sul mercato.

Attualmente la situazione è abbastanza complicata: per le prove dei vari prototipi, alcuni stati hanno già realizzato i loro "test" e rilasciato speciali licenze, ma al momento non c'è nessuna cooperazione tra le diverse giurisdizioni. In aggiunta sono già presenti delle leggi che sembrano bloccare lo sviluppo di veicoli di livello 4-5. Ad esempio la Convenzione di Vienna [10] impone la presenza del guidatore in qualsiasi movimento del veicolo (Art. 8), definendo il guidatore come *la persona che guida il veicolo a motore su strada* (Art. 1). Lo stesso discorso riguarda il settore dei trasporti dove le regolamentazioni (EC) 561/2006 e (EEC) 3821/85 [11] impongono un numero massimo di ore, per i conducenti dei veicoli.

### 4.2.3 Vantaggi

I vantaggi apprezzabili fin da subito sono legati alla mobilità dei giovani, degli anziani e dei disabili.

Potrebbero esserci impatti sullo scenario tipico moderno, dove ognuno di noi possiede un proprio veicolo. Il car sharing in questo contesto potrebbe far da padrone: la riduzione del numero di auto comporterebbe un numero minore di emissioni e maggior spazio per i parcheggi.

I tempi di viaggio potrebbero scendere drasticamente: le auto autonome e connesse possono infatti sfruttare molteplici informazioni quali aggiornamenti in tempo reale sul traffico, sui lavori, sui blocchi stradali e sugli incidenti. Utilizzando questi dati sarà possibile scegliere percorsi alternativi che riducano i tempi di percorrenza in modo significativo. Se in aggiunta sono in grado di parcheggiare anche da sole, il problema della ricerca di uno spazio libero delle città sarà solo un lontano ricordo.

Lo sviluppo di questi veicoli comporterà anche la creazione di nuove aree di lavoro altamente specializzate, soprattutto per quanto riguarda il settore IT e quello automotive.

### 4.2.4 Interessi commerciali

Le compagnie automobilistiche ed i loro fornitori svolgono un ruolo fondamentale nel campo dell'innovazione. Lo sviluppo portato avanti da loro si basa sul paradigma dell'auto come mezzo personale. Per questo motivo i loro interessi sono principalmente legati allo sviluppo di sofisticate tecnologie di assistenza alla guida. Questo approccio è dettato dal bisogno principale di realizzare prodotti che abbiano un certo fascino per il consumatore.

L'industria automobilistica inglese ha avviato degli studi per comprendere quale possa essere l'impatto dei veicoli *autonomi* e *connessi* sul mercato.

I grandi colossi tedeschi stanno promuovendo le auto *driverless* e al momento stanno effettuando numerosissimi test per perfezionarle. In particolare Audi, BMW e Mercedes Benz sono molto attive in questo settore e stanno conducendo esperimenti in tutto il mondo. L'interesse per la guida autonoma è confermato anche dall'acquisizione di **Here** - la divisione Nokia per i servizi di localizzazione - da parte di Audi, BMW e Daimler per 2.8 miliardi di Euro. Un'altra prova significativa è stata la collaborazione tra BMW e l'azienda tecnologica cinese Baidu per la realizzazione di sistemi di aiuto alla guida.

Il gruppo Volvo e Scania, i leader nel mercato svedese, hanno mostrato fin da subito particolare interesse per la completa automazione sia dei veicoli privati sia di quelli destinati al trasporto merci. Già dal 2009 Volvo ha partecipato al progetto **SARTRE** - nato e finanziato dalla Comunità Europea ai fini di realizzare tecnologie per la sicurezza in ambito stradale - mentre Scania è il partner principale di COMPANION - il progetto successore di SARTRE.

In Francia, Renault e il gruppo PSA stanno sviluppando dei meccanismi per la comunicazione *vehicle-to-vehicle (V2V)* e *vehicle-to-infrastructure (V2I)*. La Renault sta anche sviluppando un nuovo modello di automobile senza guidatore, la *Next Two*, che lancerà nel mercato entro il 2020.

La situazione è un po' diversa negli Stati Uniti: nonostante la forte spinta rivoluzionaria seguita da Google e Uber, le compagnie automobilistiche, quali Tesla e Delphi, puntano ad un approccio più graduale, concentrandosi sullo sviluppo di veicoli altamente automatizzati ma non completamente autonomi (livello 3-4).

## 4.3 Automazione in agricoltura

Sebbene siano stati investite grosse quantità di denaro sui veicoli a guida autonoma, il mercato automobilistico si è focalizzato sulle soluzioni legate alle strade cittadine. I prototipi realizzati finora sfruttano elementi tipici delle carreggiate quali strisce bianche, segnaletica orizzontale e verticale, birilli, coni e paletti. Gli algoritmi implementati si appoggiano alla connessione costante fornita dalle sempre più veloci reti mobili 4G (a breve 5G) e interagiscono con le telecamere poste agli incroci dei semafori o presenti su altri veicoli intelligenti nelle vicinanze.

Il mondo dell'agricoltura è rimasto fuori da questa rivoluzione e le soluzioni proposte sono differenti da quelle formulate per i veicoli cittadini.

### 4.3.1 Tecnologie

Nel mercato dei veicoli agricoli si sono sviluppate tecnologie differenti da quelle elencate nella sezione 4.1. Negli ultimi 20 anni c'è stata una forte spinta tecnologica che ha introdotto, nell'agricoltura di livello avanzato, controlli elettro-idraulici, sensori, sonde, sistemi di posizionamento GPS e GLONASS.

In particolare il sistema GPS tradizionale non era abbastanza preciso per gli scopi dettati da questo tipo di mercato, per questo sono state studiate e sviluppate soluzioni basate su correzioni RTK [12] e ricevitori dotati di maggior precisione.

**Parallel tracking** Una delle prime soluzioni adottate è quella del *parallel tracking*: un sistema di guida che aiuta l'operatore sul veicolo ad effetturare le manovre di sterzata, fornendo dei feedback audio o visivi. In particolare, questo tipo di sistema, avverte il guidatore nel caso in cui la traiettoria del veicolo si trovi fuori il percorso desiderato. Il *parallel tracking* ha la sua massima utilità nei veicoli che presentano grandi dimensioni (realizzati per la coltivazione o per lo spargimento di concime e fertilizzanti) ed i benefici di un campo efficientemente coperto, senza troppi spazi vuoti tra le varie corsie, sono facilmente apprezzabili. Il problema principale di questo sistema è dato dalla scarsa accuratezza del sistema GPS (Figura 4.1), per questo si utilizza principalmente in situazioni che richiedono una precisione medio-bassa.

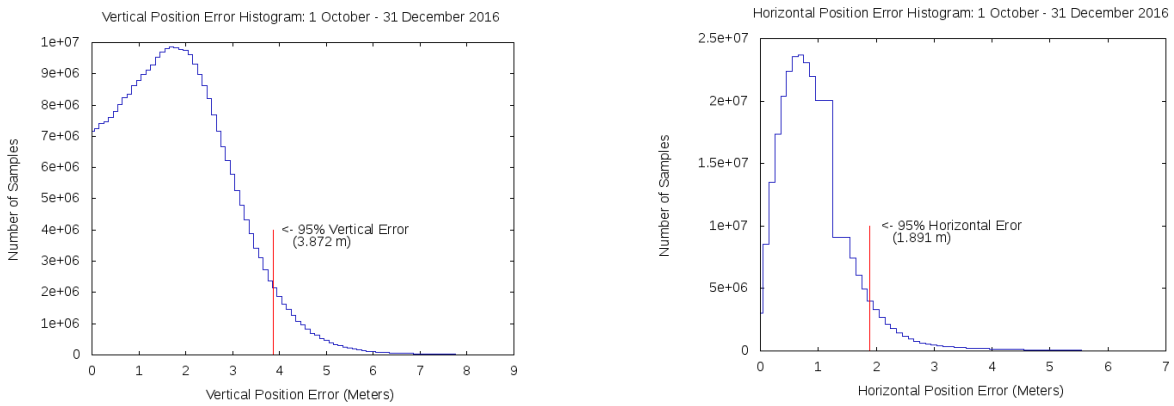



Figura 4.1. Istogrammi rappresentanti gli errori GPS rilevati da uno studio eseguito tra il 1 ottobre e il 31 dicembre 2016 (Fonte: William J. Hughes Technical Center [13]).

**Automatic guidance** Realizza un controllo in catena chiusa di un veicolo che segue una linea retta, oppure una curva, sfruttando la sterzata assistita. I componenti principali sono:

- ricevitore GPS
- sistema di sterzata
- interfaccia utente (tipicamente un display o una light bar)

Le soluzioni precedenti offrivano soltanto un aiuto visivo/acustico all'operatore, senza però poter mai intervenire direttamente. Adesso nei veicoli sono stati aggiunti sistemi di automazione di sterzata che sfruttano controllori E/H (elettro-meccanici), che si occupano di ruotare il volante e le ruote nella posizione desiderata. Uno degli aspetti legati all'*automatic guidance* è l'incremento della produttività dovuto ad operazioni più precise, ma soprattutto più veloci. Altri benefici sono legati agli operatori che sono sottoposti ad una minor fatica e minor esposizione di sostanze dannose. Tali benefici hanno incrementato di conseguenza anche la produttività degli esseri umani, che hanno più tempo a disposizione per svolgere i compiti assegnati.

Tuttavia non si tratta di un sistema di guida autonoma *driver-less*, ma solo di un "aiuto" alla guida. Gli operatori a bordo dei veicoli sono ancora presenti ed agiscono su acceleratore, freni e volante in tutte le fasi del lavoro.

**Navigation system** Per riuscire ad ottenere misure sempre più precise, si è passati dall'utilizzo del semplice GPS ad altre tecnologie:

1. **GLONASS**: è un sistema di navigazione satellitare, realizzato dall'agenzia spaziale russa. E' dotato di 24 satelliti attivi, a differenza del concorrente americano, GPS, che ne ha a disposizione 31. Quindi dispone di una copertura inferiore.

2. **GNSS** - *global navigation satellite system*: con questo termine si intende la navigazione satellitare che sfrutta diversi sistemi, quali GPS, GLONASS, Galileo e Beidou, ai fini di avere una maggior copertura e precisione.

3. **RTK**: è l'acronimo di Real Time Kinematic. Una rete RTK è composta da un insieme di stazioni GNSS permanenti che generano, attraverso diversi algoritmi, delle correzioni da inviare al dispositivo.

Combinando l'utilizzo di GPS e GLONASS, gli errori sono scesi nell'ordine dei 30cm; utilizzando anche le correzioni RTK si è arrivati fino a 1cm [14].

Il livello di precisione necessario però non è sempre uniforme, ma è fortemente legato allo scenario e ai differenti task agricoli. Ovviamente a maggior precisione è legato un costo maggiore. La tolleranza di errori fino ad 1 metro (solo GPS) è utile, ad esempio, nell'ambito della mappatura dei terreni, per ottenere informazioni sui livelli di azoto o di altre sostanze presenti. Una precisione che arriva fino ai decimetri (GPS + GLONASS) è legata all'utilizzo di sistemi di guida per la coltivazione e per la diffusione di fertilizzanti e concime. Nell'agricoltura di precisione con macchine a dosaggio variabile, si sfrutta il GNSS con correzioni RTK e sistemi sensoriali quali camere e laser lidar, per ottenere un'accuratezza molto elevata (~1cm).

**Machine-to-machine** Lo scambio di informazioni tra veicoli permette una miglior e rapida documentazione di tutto ciò che è avvenuto in un terreno. Sistemi

telematici offrono la possibilità di comunicare direttamente "all'ufficio" importanti notizie, rappresentando uno strumento necessario per il processo di gestione e strategico di un'azienda di questo settore.

### 4.3.2 Sistema di guida autonoma dei veicoli agricoli

Il mondo dell'agricoltura ha seguito un percorso parallelo rispetto a quello automobilistico. La spinta tecnologica degli ultimi anni ha permesso di ottenere dei risultati significativi dal punto di vista dell'efficienza lavorativa. I dati raccolti in questi anni hanno contribuito alla realizzazione di una *storia* dei terreni di lavoro, potendo così realizzare delle strategie che hanno portato la produttività a livelli altissimi. Tuttavia non c'è stata nessuna grande azienda, al pari di Google, che abbia avuto il coraggio di mettersi in gioco per realizzare sistemi totalmente autonomi.

La tecnologia del Fog Computing potrebbe essere un buon punto di partenza per sviluppare soluzioni *driverless*: l'intelligenza presente su un singolo nodo è in grado di gestire un sistema complesso, come quello della guida autonoma, riuscendo ad essere indipendente dal Cloud.

### 4.3.3 Vantaggi del Fog Computing

Gli algoritmi di guida autonoma per un veicolo agricolo, che si muove ad una velocità relativamente molto bassa, possono permettersi tempi di reazione decisamente maggiori rispetto a quelli utilizzati per le automobili; ed inoltre risultano essere molto meno complessi. Questo permette la loro esecuzione anche in locale, direttamente dove serve. Una versione più complessa di tali algoritmi può contemporaneamente girare sul Cloud, al fine di fornire maggior supporto al singolo nodo durante il processo decisionale. L'utilizzo del Fog Computing permette altresì l'operabilità di questi veicoli anche nei contesti in cui la connettività risulta essere limitata o assente, riduce il numero di informazioni da inviare al Cloud e ha a suo vantaggio una latenza veramente molto bassa. L'invio dei soli dati significativi si traduce in ***elevata scalabilità***: supponiamo di avere una fattoria con 10 trattori e che il loro complesso sistema di guida si trovi sul Cloud, in questo caso il flusso di informazioni provenienti dai sensori (camere, radar, gps, ecc...) deve essere trasmesso tramite la rete Internet; se per aumentare la produttività la fattoria acquistasse altrettanti trattori, il numero delle informazioni da inviare al Cloud raddoppierebbe, provocando quindi congestione sulla rete. Col Fog Computing invece i dati rimangono dove servono e la rete viene sfruttata solo allo scopo di monitoraggio o di supporto in particolari circostanze.

Altro vantaggio importante riguarda la rimozione dell'operatore a bordo del veicolo: la riduzione del peso e la realizzazione di veicoli più piccoli comporta una maggiore produttività; gli operatori non si trovano più esposti a situazioni climatiche critiche o a sostanze dannose; infine un unico operatore specializzato si può

occupare di gestire decine e decine di trattori contemporaneamente stando seduto a chilometri di distanza dal terreno.

### 4.3.4 Use case: trattore *self-driving*

L'idea è quella di realizzare un sistema di guida autonoma che riesca a controllare un veicolo, che simuli il comportamento di un trattore, in base alle informazioni ricevute localmente dai sensori. Le decisioni possono essere tuttavia influenzate anche da fattori esterni, che vengono forniti da applicazioni più complesse che girano sul Cloud. Il caso d'uso proposto prevede la presenza di due diversi tipi di algoritmi.

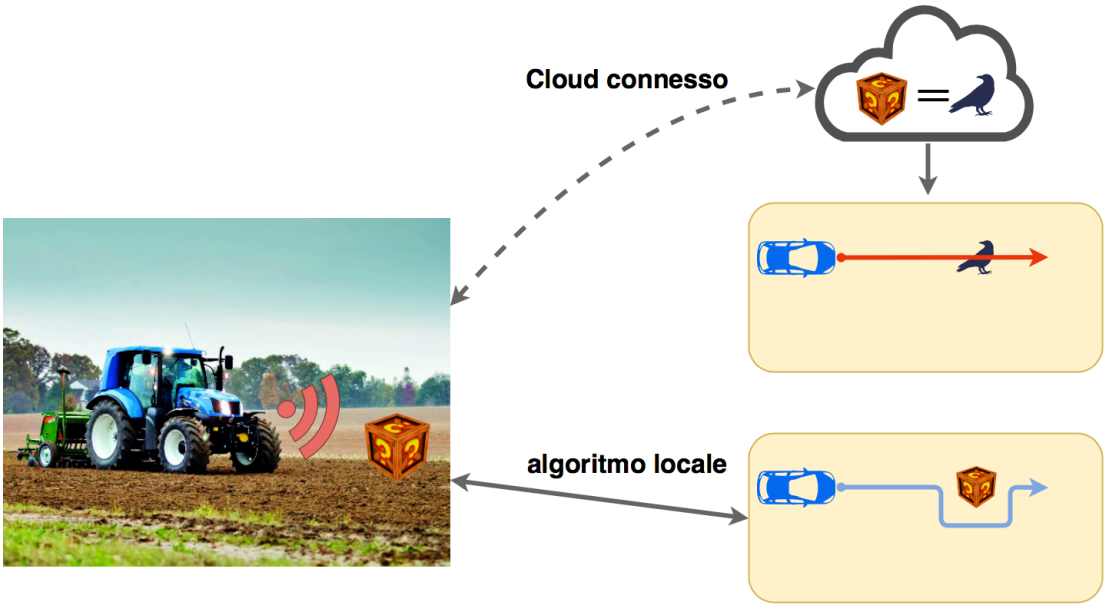



Figura 4.2. Caso d'uso: trattore con guida autonoma.

Il primo è quello che risiede in locale sul nodo Fog e si occupa di:

- inviare i comandi di guida
- monitorare lo stato dei sensori
- seguire un percorso prestabilito
- intraprendere le dovute azioni in caso di ostacoli
- comunicare col Cloud ed inviare dati significativi

Il secondo, invece, è quello che risiede lontano dal trattore. Ha quindi una latenza maggiore rispetto al primo, ma possiede risorse di storage e computazionali maggiori rispetto a quelle del singolo nodo Fog. Svolge le funzioni di:

- analizzare e comprendere il tipo di ostacolo
- inviare informazioni al sistema di guida che risiede sul nodo

Come mostrato in Figura 4.2, l'algoritmo locale esegue l'azione di evitare ogni singolo ostacolo trovi nel cammino, qualunque esso sia. Nel caso in cui la connettività verso il Cloud fosse presente, è possibile inviare dei dati significativi (ad esempio dei frame catturati da una fotocamera) per analizzare il tipo di ostacolo incontrato e capire se si tratta di una "minaccia" o meno. Ad esempio se un uccello si trova sulla stessa direzione del trattore, è preferibile che quest'ultimo non cambi il percorso stabilito, ma continui con il proprio lavoro "ignorando" l'ostacolo che ha di fronte. L'algoritmo che risiede nel Cloud si occupa appunto di informare quello locale segnalando la non pericolosità dell'animale che ha di fronte.

Date le grandi dimensioni di un trattore, si è deciso di realizzare una dimostrazione utilizzando un rover. Data la grandezza di quest'ultimo, non è stato possibile posizionare il nodo Fog a bordo del veicolo stesso, quindi i due comunicano con una connessione WiFi. La configurazione di alto livello di questo sistema viene mostrata in Figura 4.3 e verrà successivamente analizzata in dettaglio nel Capitolo 6.

E' importante notare che il sistema di guida autonoma sia composto da diverse parti, ognuna con un compito specifico, che lavorano sinergicamente per fornire le giuste istruzioni al rover. Il sistema istanziato sul Fog è facilmente controllabile da remoto. Se un determinato veicolo deve seguire un altro tipo di task o lavorare su un altro terreno, basta semplicemente bloccare l'applicazione in esecuzione sul nodo per caricarne una versione differente con i lavori da eseguire. Il provisioning in questo caso risulta essere comodo e veloce. E' possibile inoltre accedere al nodo dall'esterno per controllare lo stato delle applicazioni e visualizzare, in tempo reale, i log del sistema di riconoscimento ostacoli e di guida autonoma.

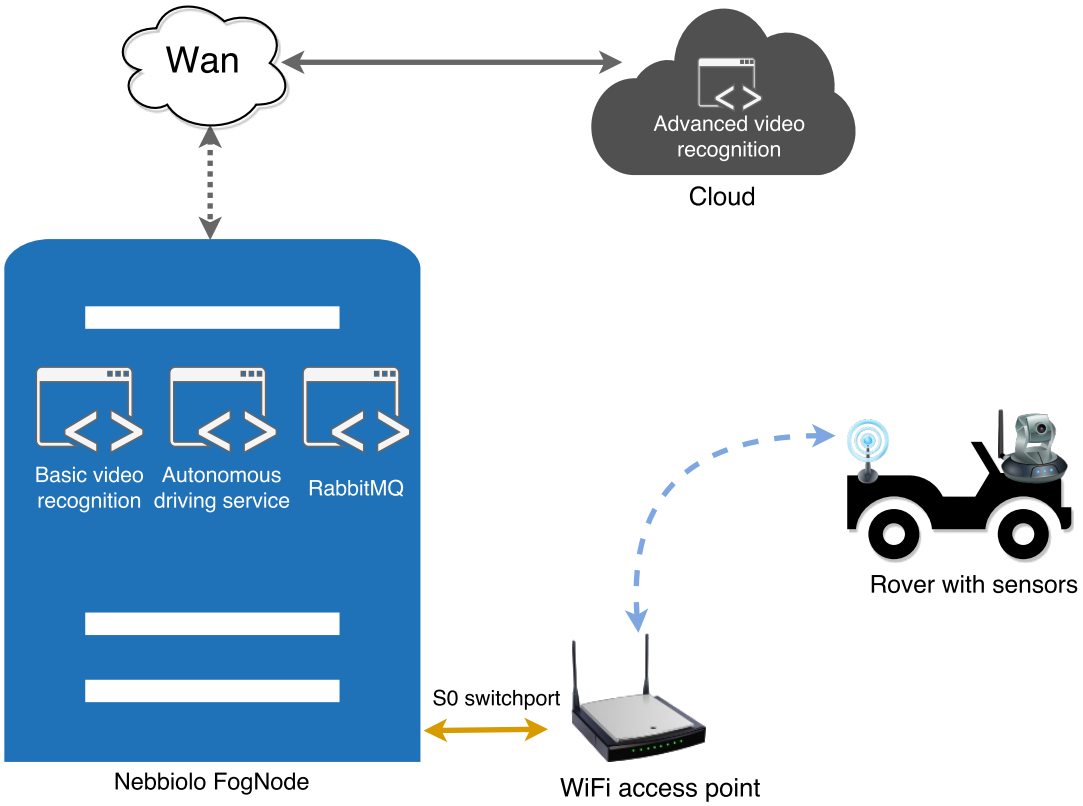



Figura 4.3. Configurazione Rover e Nebbiolo fogNode.

# Capitolo 5

# Strumenti utilizzati

Diversi strumenti software sono stati utilizzati per la realizzazione del sistema di guida autonoma, che verrà presentato ed esaminato nel Capitolo 6. Prima di entrare nel dettaglio implementativo è dunque importante esaminare tali applicativi. Molte sezioni di questo capitolo sono fortemente ispirate ai siti e alle documentazioni degli stessi.

## 5.1 Docker

Docker è una delle piattaforme software più famose al mondo che permette di eliminare il problema "*sulla mia macchina funziona*" quando si collabora con altre persone sulla realizzazione di un applicativo. Sviluppatori, operatori ed aziende utilizzano docker per risolvere i problemi di distribuzione delle applicazioni. Il progetto è open source e si basa sull'utilizzo dei **container** [15].

### 5.1.1 Container

Un container è un pezzo di software che include al proprio interno tutto quello di cui ha bisogno per essere eseguito: codice, tool di sistema, librerie e settaggi. I software che vengono racchiusi in un container possono essere eseguiti sia su Linux che su Windows e MacOS e, nonostante ciò, funzioneranno sempre allo stesso modo. Sono quindi indipendenti dall'ambiente in cui vengono istanziati e questo riduce, come accennato prima, i problemi che nascono all'interno dei team quando si utilizza software differente sulla stessa infrastruttura.

L'idea dei container non è nuova, ma la si può vedere come una naturale evoluzione dei *chroot* e delle FreeBSD *jails* disponibili già da molto tempo [16]. In passato altre soluzioni commerciali, quali Virtuozzo/OpenVZ e LXC, avevano provato ad utilizzare questo tipo di tecnologia, ma senza molto successo. Bisogna fare

attenzione però: un container non è una macchina virtuale, anzi le due entità sono totalmente differenti (Figura 5.1).

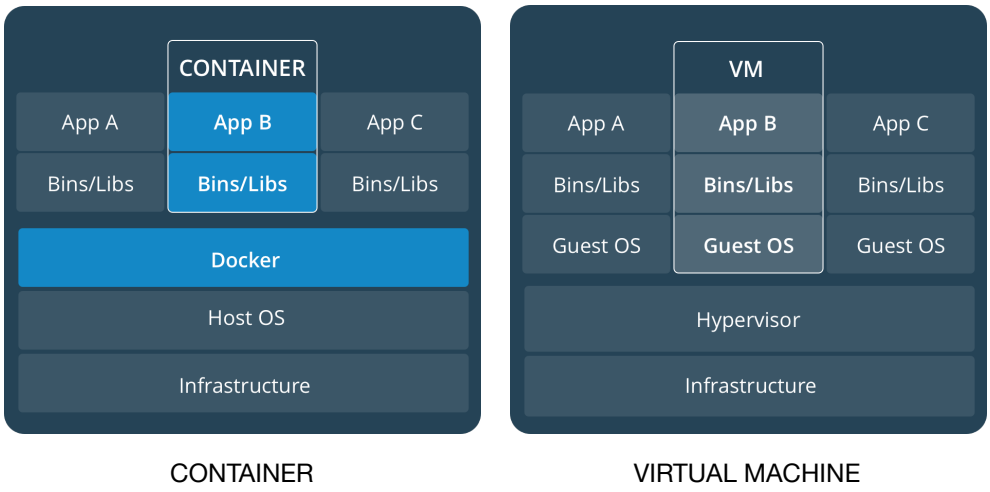



Figura 5.1. Differenze tra container e virtual machine (Fonte: Docker [15]).

**Container vs Virtual Machine** Container e VM, seppur sembrino molto simili tra loro e sfruttino gli stessi benefici legati all'isolamento delle risorse, nella sostanza risultano essere molto diversi. Detto in termini semplici: un container virtualizza un sistema operativo, mentre una virtual machine l'hardware. Per questo motivo i primi sono molto più efficienti dei secondi.

Un container è un'astrazione del layer applicativo che combina in un'unica soluzione codice e dipendenze. Più container possono essere istanziati sulla stessa macchina e condividere contemporaneamente lo stesso kernel, ognuno di essi viene infatti eseguito come un singolo processo in *user space*. I container hanno bisogno ovviamente di minor spazio delle VM ed il loro avvio avviene quasi immediatamente.

Con Virtual Machine si intende un'astrazione fisica dell'hardware. C'è la necessità di avere un hypervisor che si occupi di gestire più VM in una sola macchina. Ogni VM include una copia **intera** del sistema operativo, una o più applicazioni e tutte le librerie necessarie, per un totale di decine di GB. Il boot è molto spesso lento e non paragonabile ai tempi di avvio di un container.

### 5.1.2 Networking

Un altro aspetto importante di Docker è quello legato al networking. Durante il primo avvio, vengono create automaticamente tre reti: *bridge*, *none* e *host* [17]. Nel

momento in cui si manda in esecuzione un container, è possibile selezionare tramite il comando `-network` quale rete utilizzare.

- **bridge**: è la rete di default. Se non specificato diversamente, tutti i nuovi container vengono collegati a questa rete. E' associata all'interfaccia virtuale `docker0` e gli IP assegnati sono del tipo 172.17.0.1/16. Conoscendo l'IP di un altro container, è possibile comunicare con esso.

- **none**: quando un container viene collegato a questa rete non riceve alcun indirizzo IP se non quello di loopback. Non può accedere alla rete esterna e a nessun altro container. In pratica viene totalmente isolato.

- **host**: permette ai container ad essa collegati di condividere tutto lo stack network dell'host. Tutte le interfacce dell'host saranno quindi visibili dal container.

- **custom**: è possibile creare delle nuove reti, assegnandone nomi e range di indirizzi personalizzati. I container che vengono collegati alle reti *personalizzate* possono comunicare tra loro conoscendo semplicemente il nome del container, non servono in questo caso gli indirizzi IP. Questo è merito del server DNS interno che il demone di Docker lancia alla creazione della rete definita dall'utente. Se la risoluzione dei nomi non va a buon fine con il server DNS locale, allora vengono contattati direttamente i server DNS esterni.

Per la mappatura delle porte verso l'esterno esistono due keywords, `EXPOSE` e `PUBLISH`:

- **EXPOSE**: serve ai fini di documentazione. L'utilizzo di questa keyword è pertanto opzionale e si inserisce del Dockerfile (file utile per la creazione di un container) per indicare a chi manderà in esecuzione il container quali porte pubblicare verso l'esterno.

- **PUBLISH**: utilizzando "publish" è possibile demandare direttamente a Docker l'apertura delle porte richieste nel momento in cui viene istanziato il container. Generalmente Docker sceglie una porta a random con un valore maggiore di 3000. E' possibile richiedere il mapping con una porta specifica dell'host, ma in questo caso bisogna fare attenzione perché tale porta potrebbe essere già occupata da qualche altro processo.

## 5.2 RabbitMQ

RabbitMQ è un message broker open-source che implementa il protocollo *Advanced Message Queue Protocol (AMQP)* per lo scambio di dati tra processi, applicazioni

e server. Scritto in Erlang, si basa sul framework *Open Telecom Platform (OTP)* e fornisce librerie client per diversi linguaggi di programmazione [18].

All'interno di RabbitMQ possono essere definite diverse *code*, alle quali le applicazioni hanno la capacità di connettersi per scambiare messaggi. Un **messaggio** può includere qualsiasi tipo di informazione per poter sincronizzare due o più applicazioni. Questi non vengono direttamente spediti nelle code, ma devono essere gestiti dagli **exchange**. Un exchange si occupa del routing dei messaggi: dopo averli prelevati dall'applicazione producer, li smista all'interno delle code grazie all'aiuto delle *routing key*. Ecco come funziona il flusso di messaggi con RabbitMQ:

1. Il produttore pubblica il messaggio verso un exchange (ne esistono diverse tipologie, bisogna quindi specificare di quale si tratta).

2. L'exchange, non appena ricevuto il messaggio, ha la responsabilità di gestirlo correttamente e può farlo sfruttando gli attributi contenuti al suo interno.

3. Devono essere creati adesso dei link - *binding* - dall'exchange alle code. In Figura 5.2 è possibile vedere come uno stesso exchange può essere collegato a più code. Questo non è un problema: gli attributi interni al messaggio permettono all'exchange di sapere quale coda scegliere senza commettere errori.

4. I messaggi restano in coda fin quando non vengono "prelevati" da un consumatore.

5. Il consumatore adesso può finalmente gestire il messaggio ricevuto.

### 5.2.1 Tipologie di Exchange

Come accennato precedentemente, esistono diverse tipologie di exchange ed i client possono utilizzare quelli predefiniti o crearne di nuovi. Le varie tipologie sono:

- **direct**: i messaggi vengono consegnati direttamente alla coda specificata nella routing key. Le *routing key* sono degli attributi inseriti nell'header del messaggio da parte del producer. Possono essere viste come gli indirizzi che l'exchange utilizzerà per decidere la destinazione. La o le code prescelte sono quelle in cui la *binding key* combacia esattamente con la routing key. Di default è presente un exchange di tipo direct che prende i nome di `amq.direct`.

- **default**: è un exchange di tipo direct senza alcun nome, generalmente si fa riferimento ad esso con una stringa vuota. Un messaggio che viene inviato a questo exchange verrà consegnato alla coda il cui nome è uguale a quello della routing key. Tutte le code vengono automaticamente collegate a questo exchange.

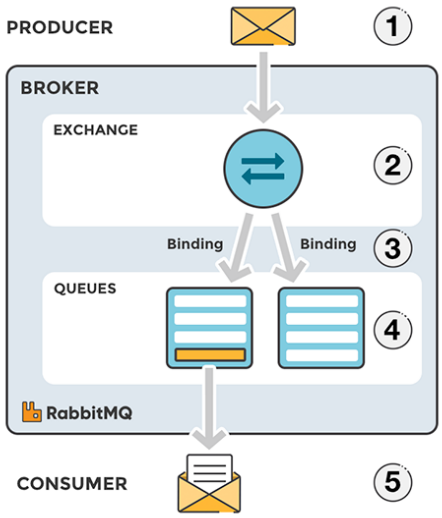



Figura 5.2. Flusso di messaggi in RabbitMQ (Fonte: CloudAMQP [18]).

- **topic**: effettua il routing in base alle parole chiavi inserite nella routing key e nelle binding key. La routing key deve essere una lista di parole, separate da un punto "."; all'interno possono esserci anche degli asterischi "" per poter specificare che in quella posizione va bene qualsiasi parola presente nella binding key. Ad esempio: routing key = `example.*.*.b.*` – in questo caso il messaggio verrà consegnato a tutte quelle code la cui binding key ha come prima parola "example" e come quarta parola "b". E' presente anche un altro simbolo particolare, il cancelletto "#", che viene utilizzato per indicare che possono essere presenti anche altre parole. Si consideri ad esempio la routing key "polito.it.#" , tutte le binding key che iniziano per "polito.it" soddisferanno il matching.

- **fanout**: questo exchange copia e inoltra il messaggio ricevuto a tutte le code ad esso collegato. La routing key viene semplicemente ignorata, tutte le code connesse vanno bene. Non ha quindi bisogno di alcun meccanismo di matching, quello che viene scritto nella routing key dal produttore non ha nessuna valenza. Questo exchange risulta essere molto utile e veloce nel caso in cui lo stesso messaggio deve essere trasmesso a una o più code nelle quali i consumatori processano i dati ricevuti in maniera indipendente. Ne è presente uno di default che prende il nome di `amq.fanout`.

- **headers**: questa tipologia è molto simile a quella del *topic*, ma sfrutta gli argomenti presenti sull'header del messaggio invece della routing key. Un messaggio viene inoltrato in una determinata coda se gli argomenti dell'header

combaciano con quelli del binding. E' quindi possibile avere più argomenti. Uno in particolare, che prende il nome di `x-match`, indica se a combaciare siano tutti gli argomenti o va bene anche soltanto uno di essi. Può avere due valori differenti: *all* (di default) oppure *any*. Se nulla viene specificato, allora tutti gli argomenti saranno obbligatori; con la presenza di *any* ne basta soltanto uno per dare l'ok al sistema di matching. Anche in questo caso è possibile trovarne uno creato di default: `amq.headers`.

- **dead letter**: in tutti i casi precedenti, se nessuna coda rispetta i requisiti richiesti, allora il messaggio viene scartato silenziosamente. Ci sono degli scenari in cui questo non va bene, per tale ragione RabbitMQ fornisce quest'ultimo tipo di exchange che si occupa di recuperare i messaggi che non sono stati consegnati a nessuno (detti anche *"lettere morte"*).

### 5.2.2 Sistema di gestione

Il sistema di gestione, chiamato *RabbitMQ management*, fornisce un'interfaccia user-friendly che permette di monitorare e gestire il server RabbitMQ direttamente da un browser web. Code, connessioni, exchange, canali e molto altro possono essere gestiti direttamente da qui. E' possibile anche monitorare il rapporto di messaggi inviati e ricevuti oppure scambiare messaggi manualmente attraverso l'interfaccia web. Il RabbitMQ management è di fatto un plugin che sfrutta le API HTTP di RabbitMQ fornendo informazioni fondamentali in caso di debug o controllo del sistema.

Nella schermata principale sono presenti due grafici che mostrano in tempo reale il numero di messaggi in coda e la velocità con cui questi arrivano. Sono presenti anche altri campi che danno informazioni più dettagliate:

- **ready**: indica il numero di tutti i messaggi che sono pronti per essere consegnati nelle varie code.
- **unacked**: i messaggi, nel momento in cui vengono inviati dal server, devono essere "confermati" tramite l'invio di un *ack*. Non possono essere cancellati dalla memoria di RabbitMQ fino a quando l'ack non è stato ricevuto.
- **total**: la somma dei due campi precedenti.

Un'importante sezione è quella che riguarda le connessioni e i canali.

**Connessione e canale** Per *connessione* si intende una tradizionale connessione TCP tra l'applicazione e il server di RabbitMQ; il termine *canale* invece indica una connessione virtuale che si instaura dentro una connessione TCP. Connessioni e canali possono troversi in diversi stati: starting, tuning, opening, running, flow, blocking, blocked, closing, closed. Ad esempio se un client sta trasmettendo troppi

dati e supera spesso la soglia massima, allora lo stato della connessione in questo caso diventa *flow* e RabbitMQ si occupa in qualche modo di limitarlo.

**Controllo del flusso** A partire dalla versione 3.3 di RabbitMQ è possibile scovare i colli di bottiglia e prendere delle contromisure per evitare le congestioni delle code. Una prima versione del controllo del flusso era presente già dalla versione 2.8 ed inseriva nello stato *flow* tutte le connessioni che stavano generando troppi messaggi. Il problema principale è che, all'interno della stessa connessione TCP, il client può inviare messaggi che finiscono su code diverse ed è quindi impossibile sapere quale sia la coda troppo piena. Per questo motivo, con le nuove versioni, è stata effettuata la scelta di separare la connessione dal canale [19]. Gli scenari possibili sono:

1. La connessione è in stato di *flow*, ma nessuno dei suoi canali lo è: questo significa che uno o più canali sono la causa della congestione. Probabilmente troppi e piccoli messaggi arrivano al server e la CPU è impegnata con il meccanismo del routing.

2. La connessione è in stato di *flow* e alcuni suoi canali lo sono, ma nessuna coda sta pubblicando: anche in questo caso uno o più canali sono la causa. Le situazioni possono essere due: o il server è a corto di CPU per via del routing oppure ci sono troppe operazioni di I/O su disco. Succede spesso se si spediscono moltissimi piccoli messaggi persistenti.

3. La connessione, alcuni canali e alcune code sono nello stato di *flow*: il message store è la causa della congestione. La velocità del disco del server non basta per scrivere i messaggi. E' il tipico caso di messaggi molto grandi e persistenti.

Per merito della distizione tra canale e connessione è adesso possibile trovare e risolvere le cause di congestione. In aggiunta è presente un ulteriore strumento, il *consumer utilisation*, che permette di capire se e quando il client, a causa di una congestione di rete, non riesce a ricevere tutti i messaggi ad esso diretti.

## 5.3 OpenCV

OpenCV è una libreria video rilasciata sotto licenza BSD utilizzabile sia per scopi accademici che per quelli commerciali. Ha interfacce C++, C, Python e Java e supporta Windows, Linux, MacOS, iOS e Android. In questa tesi è stata utilizzata la versione 3.2 per il linguaggio Python su Linux. Nasce per essere efficiente dal punto di vista computazionale e permette il suo utilizzo anche in ambiti real-time. Utilizzando OpenCL è possibile inoltre sfruttare i vantaggi dell'accelerazione hardware.

OpenCV è composta da vari moduli che possono contenere al loro interno diverse librerie statiche o condivise. I più importanti sono:

1. **Core functionality**: ingloba tutte le strutture dati e le funzioni base utilizzate dagli altri moduli.
2. **Image processing**: include filtri lineari e non lineari, trasformazioni geometriche delle immagini, conversione dello spazio del colore, istogrammi e molto altro.
3. **Video**: al suo interno presenta i moduli per l'analisi video, la motion compensation, eliminazione dello sfondo e algoritmi per il tracciamento degli oggetti.
4. **Video I/O**: codec per la cattura e l'elaborazione video

I paragrafi seguenti faranno riferimento a funzioni e strutture di OpenCV per C++ poiché è presente maggior documentazione al riguardo.

### 5.3.1 Gestione delle immagini

All'interno del modulo "Core" sono definite, come detto in precedenza, le strutture dati utilizzate da OpenCV. In particolare, all'interno di `cv.h`, è presente la classe **Mat**, fondamentale per la gestione delle immagini. *Mat* deriva dalla classe *CvArr* e rappresenta un array numerico n-dimensionale. Può essere utilizzata per la memorizzazione di valori reali o complessi di vettori e matrici, immagini in scala di grigi o a colori, vettori di campi e istogrammi [20].

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <opencv/cv.h>
4  #include <opencv/highgui.h>
5
6  int main(int argc, char* argv[])
7  {
8      // caricamento immagine da file
9      Mat* img = (Mat*)cvLoadImage( "immagine.jpg" );
10     if (!img) {
11         cout << "Errore immagine" << endl;
12         exit(-1);
13     }
14
15     // creazione nuova immagine da zero
16     Mat* img_zero = (Mat*)cvCreateImage(cvSize(100,
           100), IPL_DEPTH_8U, 3);
17
18     return 0;
19 }
```

Per caricare all'interno di questa struttura un'immagine, si utilizza la funzione **cvLoadImage()** specificando come argomento il file da passare. Con la **cvCreateImage()** è possibile creare un'immagine senza che questa sia presente da qualche parte in memoria, basta specificare dimensione, profondità e numero di canali. E' importante sottolineare il fatto che OpenCV gestisce ogni immagine come se fosse un array monodimensionale. Nel caso di Python, invece della classe Mat si utilizza *NumPy*; fondamentalmente le operazioni che si eseguono sono identiche.

### 5.3.2 Cattura video

Analogamente a quanto fatto per le immagini, anche per i video bisogna utilizzare una struttura presente nel modulo Core: **CvCapture**.

```
1  #include <stdio.h>
2  #include <opencv/cv.h>
3  #include <opencv/highgui.h>
4
5  using namespace std;
6
7  int main( int argc, char** argv )
8  {
9      CvCapture *capture = cvCaptureFromAVI(
           "input_video.avi" );
10     if (capture == NULL) {
11         cout << "Loading error!" << endl;
12         exit(-1);
13     }
14
15     CvCapture *webcam = cvCreateCameraCapture(0);
16     if (webcam == NULL) {
17         cout << "Webcam not ready!" << endl;
18         exit(-1);
19     }
20
21     cvReleaseCapture(&capture);
22     cvReleaseCapture(&webcam);
23     return 0;
24 }
```

Con la funzione **cvCaptureFromAVI()** viene caricato il video passato come argomento della funzione all'interno di questa struttura. E' possibile acquisire anche video in tempo reale dalle videocamere collegate al proprio pc. La funzione **cvCreateCameraCapture()** serve proprio a questo scopo e prende come argomento

un intero che identifica il dispositivo di acquisizione video. Nel caso di un solo dispositivo, ad esempio la videocamera interna del laptop, bisogna passare il numero 0.

### 5.3.3 Edge Detection

Con Edge Detection si intende il riconoscimento, all'interno di un'immagine, di punti in cui la luminosità cambia in maniera repentina. Generalmente ad un cambio repentino di luminosità è associato il contorno di oggetti. Molto spesso, però, possono essere presenti rumore o fonti di illuminazione secondarie che inquinano la scena. L'algoritmo di Edge Detection utilizzato in OpenCV è quello sviluppato dall'australiano John F. Canny nel 1986. Si basa fondalmente su tre criteri:

1. Riconoscimento dei contorni principali.
2. L'individuazione di un contorno deve avvenire una e una sola volta.
3. I bordi trovati devono approssimare al meglio i contorni reali.

Esiste a tal proposito una funzione che matematicamente esprime questi tre criteri e presenta delle similitudini rispetto alla derivata prima di una funzione gaussiana. La discontinuità di luminosità può essere infatti rappresentata dai punti di massimo di una funzione.

Una funzione di questo tipo viene però facilmente disturbata dal rumore, che è un elemento quasi sempre presente nelle immagini. Per questa ragione l'algoritmo di Canny non si limita ad applicarla direttamente ma effettua delle operazioni preliminari. Innanzitutto applica il filtro gaussiano di *smoothing* per cercare di eliminare dall'immagine quanto più rumore possibile. Successivamente calcola la derivata prima della luminosità. In seguito si analizza tutti i punti con valori "elevati" e scarta quelli che sono non di massimo all'interno di un certo intorno. Infine, confronta i punti rimasti con delle threshold, una bassa ed una alta, e procede come segue: non considera i punti più bassi della soglia bassa come contorno, mentre etichetta come punti di edge quelli che sono più alti rispetto alla soglia più alta. Per quelli compresi tra le due soglie analizzerà i pixel adiacenti e, se questi sono oltre soglia massima, allora considererà anch'essi come parte del contorno.

All'interno di OpenCV, l'algoritmo di Canny è stato implementato nella funzione **cvCanny()**.

```
1 #include <opencv/cv.h>
2 #include <opencv/highgui.h>
3
4 using namespace std;
5
```

```
6  int main(int argc, char** argv) {
7      Mat* src = (Mat*) cvLoadImage("image.jpg");
8      Mat* dst = (Mat*) cvCreateImage(cvGetSize(src), 8,
           1);
9
10     cvCanny(src, dst, 50, 200, 3);
11     cvNamedWindow("Source", 1);
12     cvShowImage("Source", src);
13     cvNamedWindow("After Canny", 1);
14     cvShowImage("After Canny", dst);
15     cvWaitKey(0);
16 }
```

La funzione *cvCanny()* prende in ingresso cinque parametri: immagine sorgente, immagine destinazione, valore della soglia bassa, valore della soglia alta e un parametro che serve per la regolazione della maschera di Sobel utilizzata per ottenere la derivata prima della luminosità. In Figura 5.3 è possibile vedere un esempio di applicazione dell'algoritmo di Canny.

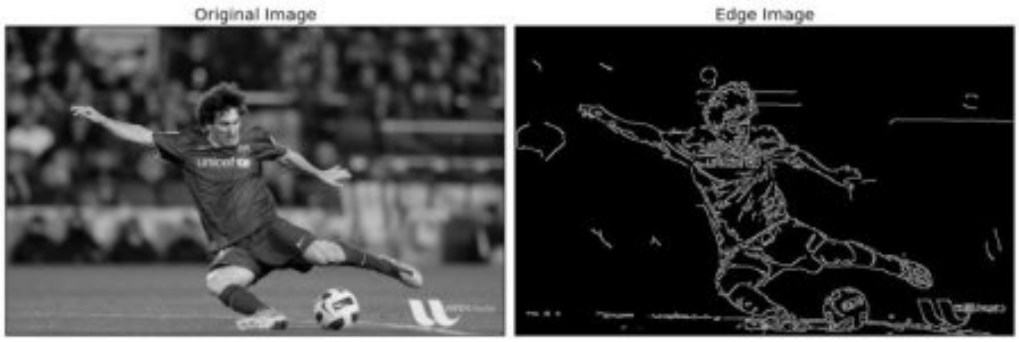



Figura 5.3. Applicazione dell'algoritmo di Canny (Fonte: OpenCV [21]).

Tale funzione risulta essere molto importante in questo lavoro di tesi, nel capitolo successivo verrà infatti esaminato il sistema di riconoscimento degli ostacoli che sfrutta l'algoritmo di Canny per l'identificazione degli oggetti (*ostacoli*) sulla scena.

# Capitolo 6

# Architettura ed implementazione

In questo capitolo vengono presentati i vari componenti dell'architettura del sistema di guida autonoma (Figura 6.1). La soluzione mostrata e sviluppata in questo elaborato si avvale dell'utilizzo di un piccolo rover ai fini di simulare il comportamento di un trattore all'interno di un terreno agricolo. L'idea è quella di realizzare un prototipo di guida autonoma sfruttando sinergicamente il Fog ed il Cloud combinando bassa latenza ad elevate capacità di elaborazione.

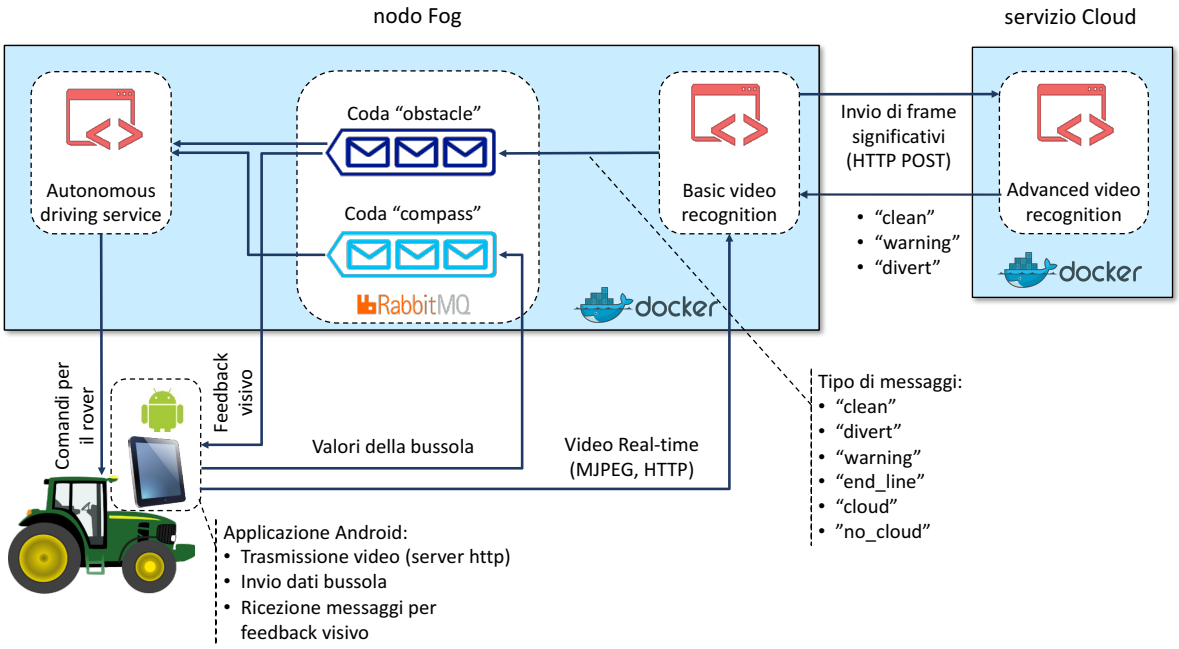



Figura 6.1. Architettura del sistema di guida autonoma.

## 6.1 Architettura generale

Per questa soluzione sono stati utilizzati:

- Rover A4WD1
- Nebbiolo fogNode NFN 300 con un modulo NFL-1000-C
- Tablet HTC Nexus 9 con Android 7.1.1
- Servizio cloud di terze parti (sloppy.io)

Nella parte superiore del rover è stato posizionato il tablet Android con la fotocamera posteriore rivolta verso il basso. Le immagini acquisite da questo device vengono trasmesse in tempo reale al fogNode tramite il protocollo HTTP; contemporaneamente i dati della bussola e dell'accelerometro vengono inviati in una coda di messaggi del message broker installato nel nodo Fog. Quest'ultimo ha in esecuzione 3 container: ***RabbitMQ***, ***Autonomous driving service*** e ***Basic video recognition***.

Il primo container è il message broker e viene utilizzato per lo scambio di informazioni tra il tablet e le varie applicazioni istanziate sul nodo fog; il secondo si occupa di inviare comandi al rover, mentre il terzo rileva gli ostacoli ed informa il sistema di guida. Infine è presente un altro container, *advanced video recognition*, che è stato mandato in esecuzione su un servizio Cloud e che comunica con il Fog attraverso la rete Internet e il protocollo HTTP.

Nei paragrafi successivi verranno esaminati in dettaglio tutti i componenti del sistema.

## 6.2 Rover

Il rover, mostrato in Figura 6.2, possiede una struttura in alluminio anodizzato che custodisce al proprio interno un pacco batterie 12.0V Ni-MH da 1600mAh e 4 motori DC (modello HN-GH12-2217Y) da 12V e 200RPM. I motori fanno girare 4 ruote dal diametro di 4.75" ed il loro controllo è affidato a BotBoarduino[23], un mix tra BotBoard II[24] e Arduino Duemilanove[25] a cui è stata aggiunta la Arduino WiFi 101 Shield[26] per ottenere connettività IEEE 802.11 b/g/n [22].

Il software a cui è stato delegato il compito di controllare il rover, effettua allo startup un processo di calibrazione della potenza dei motori, muovendo le ruote in avanti e indietro. Terminata la fase iniziale, viene effettuata una scansione delle reti WiFi: il rover si connette automaticamente all'SSID "LynxMotion", da cui ottiene i parametri di configurazione dal server DHCP. Stabilita la connessione e ricevuto l'indirizzo IP, a questo punto il sistema di controllo prova a contattare l'indirizzo 192.168.1.42 alla porta: 10002.

Figura 6.2. Immagine del Rover A4WD1 (Fonte: LynxMotion [22]).

Se tutto è andato a buon fine, il rover è pronto a ricevere i seguenti comandi:

- "I" modalità *idle/standby*
- "R" modalità *rover* (aziona i motori)
- "A" modalità *arm* (aziona il braccio) – non utilizzata in questa tesi

Quando la modalità *"rover"* è attiva, è possibile inviare i comandi per far muovere le ruote nella direzione e alla velocità desiderate. La sintassi da seguire è la seguente:

$$\boxed{\#speed\#direction*}$$

dove "speed" e "direction" sono dei valori compresi tra -100 e 100. Con il primo valore si decide il senso di rotazione delle ruote del rover (orario o antiorario), mentre il secondo serve per far ruotare il rover da sinistra a destra o viceversa. Maggiore è il valore inviato e maggiore sarà la rotazione delle ruote. E' fondamentale inviare il comando di stop, ovvero $\boxed{\#0\#0*}$, ogni qualvolta si voglia inviare una nuova stringa.

L'invio dello stesso comando può produrre risultati differenti: il movimento delle ruote dipende dal suolo e dalla carica della batteria. L'utilizzo di velocità basse (inferiori a 55) aumenta la precisione dei movimenti, ma può provocare il blocco del rover. Ad esempio è raro riuscire ad effettuare una rotazione con valori non superiori a 60 e, se questa avviene, può bloccarsi dopo poco tempo. Per questo motivo, come descritto nella sottosezione 6.3.2, il parametro che indica la velocità viene incrementato automaticamente dal sistema di guida per evitare situazioni di stallo.

## 6.3 Nebbiolo fogNode

Le 6 porte Gigabit Ethernet del fogNode sono divise in due gruppi: le prime 4 sono collegate allo switch interno del fogNode (S0, S1, S2, S3), le altre 2 sono invece dirette (E0, E1). La porta E0 viene utilizzata per fornire la connettività verso l'esterno, mentre sulla S0 è presente un access point WiFi a cui è stato assegnato l'SSID "LynxMotion".

La struttura software del fogNode è a strati: nello strato più basso è presente l'hypervisor che gestisce tutto il nodo Fog; ogni modulo (in questo caso solo uno) ha di default una AdminVM, sulla quale è possibile mandare in esecuzione container docker. Questa virtual machine, e di conseguenza tutti i container al proprio interno, possono essere gestiti sia dall'utilizzatore che da Nebbiolo. Per motivi di privacy e sicurezza, è possibile "nascondere" le attività delle proprie applicazioni creando altre VM; nel nostro caso, come spiegato nella sezione 3.3, è presente un'altra virtual machine, la OTVM1, dentro la quale sono presenti tutti i container docker utilizzati.

### 6.3.1 RabbitMQ e code di messaggi

Il primo container Docker presente dentro la OTVM1 è quello di RabbitMQ. I container che si trovano nella stessa sottorete possono contattarlo all'indirizzo 10.10.11.33, mentre chi si trova all'esterno, in questo caso il tablet, può comunicare con esso tramite l'indirizzo privato del nodo Fog 192.168.1.42. Le porte sono state mappate in modo tale da utilizzare quelle di default (porta TCP 5672 per lo scambio di messaggi e porta TCP 15672 per il management).

RabbitMQ, tramite l'utilizzo degli *exchange* (sottosezione 5.2.1), si occupa di distribuire i messaggi che riceve all'interno di particolari code, **compass** e **obstacle**.

La coda dei messaggi della bussola, o *compass*, contiene al proprio interno i valori della bussola che l'applicazione installata sul device Android invia periodicamente. Tali valori hanno un unico destinatario, l'*autonomous driving service*, e sono utili per determinare l'orientamento spaziale del rover, soprattutto in fase di inversione di marcia.

La seconda coda, *obstacle*, funziona in maniera diversa dalla prima: sono presenti più destinatari e l'exchange a cui vengono recapitati i messaggi in questo caso è di tipo *fanout*. L'autonomous driving service e l'applicazione Android sono i due destinatari, mentre il mittente è il basic video recognition. I messaggi inviati dal sistema di riconoscimento degli ostacoli possono essere di 6 tipi differenti:

- *clean*: non sono stati trovati ostacoli
- *divert*: è stato individuato un ostacolo da evitare
- *warning*: è stato trovato un ostacolo ma non rappresenta un pericolo

- *end_line*: raggiunto il limite, invertire il senso di marcia
- *cloud*: il cloud è connesso
- *no_cloud*: impossibile contattare il cloud (nessuna connessione o socket timeout)

In base a questi messaggi il sistema di guida autonoma reagisce opportunamente inviando comandi al rover. Al tablet Android queste informazioni servono soltanto per fornire un feedback visivo sullo schermo: icone con colori e forme differenti si accendono o spariscono dall'interfaccia dell'app per notificare un cambiamento di stato.

### 6.3.2 Autonomous Driving Service

Il sistema di guida autonoma, o *autonomous driving service*, è un container docker con al proprio interno un applicativo Java.

```
1  public static void main(String[] args){
2
3      /* Read information from compass queue */
4      new Thread(new Runnable(){
5          @Override
6          public void run() {
7              getCompassData();
8          }
9      }
10     ).start();
11
12     /* Read information from obstacle queue */
13     new Thread(new Runnable(){
14         @Override
15         public void run() {
16             getObstacleData();
17         }
18     }
19     ).start();
20
21     /* Start drive system */
22     drive();
23
24 }
```

Questa applicazione al suo avvio crea e manda in esecuzione due thread. Il primo si occupa di ricevere i messaggi dalla coda *obstacle*, il secondo invece da *compass*. Le funzioni `getCompassData()` e `getObstacleData()`, sfruttando il metodo **handleDelivery()** presente nell'interfaccia `com.rabbitmq.client.Consumer`, ricevono i messaggi dalle rispettive code e li salvano atomicamente all'interno di variabili thread-safe di tipo `AtomicReference<String>`.

Nel frattempo, dal thread principale, viene chiamata la funzione `drive()` che si occupa di:

1. Inizializzare tutti i flag e i parametri relativi alla velocità del rover
2. Avviare il server e mettersi in ascolto sulla porta 10002
3. Inviare i comandi di *idle* e *rover*
4. Far muovere il rover
5. Prendere le dovute contromisure in caso di ostacoli o fine del percorso

La rotazione avviene controllando in tempo reale il valore della bussola e stoppando il rover non appena sono stati raggiunti i valori richiesti. Il rover non dispone di sistemi odometrici e la sua posizione non può essere determinata dal GPS, sia per la poca precisione sia perché i test sono stati fatti in ambienti indoor. Per questo motivo il sistema di guida autonoma si basa esclusivamente sui valori ricevuti dalla bussola. Di seguito è riportata la parte di codice che serve per far ruotare il rover di 90°:

```
1  static void turn_90degree_right() throws
       InterruptedException{
2
3          // get actual compass data
4          int pos_init = position.get().intValue();
5          int pos_after = pos_init;
6          int pos_before = pos_init;
7          int speed = param;
8
9          int index = 0;
10         System.out.println("Pos_init: "+pos_init);
11         while(distance(pos_init, pos_after)<83 &&
               !rst_flag){
12             if(index%30 == 0){
13                 if(distance(pos_before, pos_after)<2 &&
                       speed<85){
14                     speed = speed+5;
```

```
15                          right(speed);
16                      }
17                      pos_before = position.get().intValue();
18                  }
19                  Thread.sleep(26);
20                  pos_after = position.get().intValue();
21                  index++;
22              }
23      os.println(STOP);
24
25  }
```

I valori presenti in questa funzione sono stati trovati sperimentalmente provando il rover direttamente sul campo. La bussola del tablet presenta infatti una certa latenza prima di segnare l'angolo corretto, i motori sono condizionati dalla carica della batteria ed il movimento del rover è legato anche al tipo di terreno/pavimentazione presente. Per questo motivo la velocità di rotazione non è fissa, ma varia in base al movimento. Ogni 26ms viene controllato l'angolo di rotazione, non appena viene raggiunto il valore desiderato, il sistema di guida blocca i motori del rover. Durante i vari test ci si è resi conto che potrebbero insorgere situazioni di deadlock, qualora la velocità del rover fosse troppo bassa per effettuare una rotazione. Per questo motivo l'algoritmo utilizzato prevede che ogni 780ms (26ms*30 cicli di while) si aumenti la velocità se il grado di rotazione è inferiore a 2 gradi.

I messaggi "warning", "cloud" e "no_cloud" vengono semplicemente ignorati da questa applicazione: gli unici comandi significativi sono quelli che indicano la presenza di un ostacolo da evitare (*"divert"*) o la fine del percorso (*"end_line"*). Nel primo caso l'algoritmo si occupa di eseguire una serie di manovre per evitare l'ostacolo per poi rimettersi nuovamente nella stessa corsia precedente; nel secondo invece si effettua una rotazione di 90°, si cambia corsia e si effettua un'altra rotazione. Alla fine il rover si troverà ruotato di 180° e nella corsia successiva: da lì potrà continuare il proprio lavoro.

Per resettare il sistema di guida, stoppare e riavviare il server, è stato aggiunto un flag che si abilita alla ricezione del messaggio "restart". Tale messaggio può essere ricevuto solo nella coda *compass* e viene inviato dall'utente tramite la pressione del tasto di restart nell'interfaccia dell'applicazione Android.

### 6.3.3 Basic Video Recognition

Il terzo container istanziato nel fogNode si basa sulla versione di Ubuntu 16.10 con in aggiunta Python 2.7.10, OpenCV 3.2, ffmpeg e gstreamer. Sono state inoltre inserite le librerie "*pika*" (implementazione Python del protocollo AMQP 0.9.1) e

"*requests*" (per comunicare attraverso il protocollo HTTP). Questo container lancia all'avvio uno script Python, il cuore del sistema di riconoscimento ostacoli.

Il primo passo è quello di instaurare una connessione con RabbitMQ ed inizializzare tutte le strutture necessarie per il riconoscimento video, quali gli array multidimensionali per la definizione del range cromatico e quelli monodimensionali per la realizzazione delle trasformazioni morfologiche [27].

Successivamente si entra in un ciclo che serve per elaborare i frame ricevuti dall'applicazione Android.

***find_obstacle()*** Ogni frame che arriva, viene mandato alla funzione *find_obstacle()* che si occupa di:

1. Convertire l'immagine in scala di grigi (l'applicazione dell'algoritmo di Canny non è possibile per le immagini a colori)
2. Applicare il filtro di sfocatura mediana (per cercare di rimuovere i disturbi nell'immagine)
3. Utilizzare l'algoritmo Canny Edge Detection (sottosezione 5.3.3)
4. Controllare la presenza di contorni all'interno dell'immagine
5. Inviare il frame alla funzione *find_end()* se sono stati trovati degli ostacoli
6. Inviare il frame alla funzione *find_redobject()* se sono stati trovati degli ostacoli, ma non si tratta dei confini del percorso
7. Inviare il frame alla funzione *send_to_cloud()* se sono stati trovati degli ostacoli, non si tratta dei confini del percorso ed hanno un colore compreso tra il rosso ed il giallo

Se nessun ostacolo viene rilevato da questa funzione, viene semplicemente pubblicato il messaggio *"clean"* sulla coda *obstacle* e lo script si mette di nuovo in attesa del prossimo frame. Quando invece un ostacolo è presente all'interno dell'area catturata dalla fotocamera del tablet, la funzione *find_obstacle()* invia una copia del frame originale alla funzione *find_end()*, alla quale è stato delegato il compito di individuare, mediante la `findChessboardCorners()`, una scacchiera di dimensioni pari a 4x4. Tale funzione è nativa di OpenCV e normalmente viene utilizzata per la calibrazione dell'angolo di visione. In questo caso, invece, il suo scopo è quello di riconoscere i confini dell'area di lavoro del rover, nei quali sono stati posti dei fogli di carta con su stampate delle scacchiere. L'utilizzo dei flag `CALIB_CB_FAST_CHECK` e `CALIB_CB_FILTER_QUADS` ha permesso di incrementare drasticamente le prestazioni della *findChessboardCorners()*.

Se i confini sono stati riconosciuti dal sistema, il messaggio *"end_line"* viene quindi pubblicato sulla *obstacle*, altrimenti si passa alla fase successiva.

***find_redobject()*** Un ostacolo è stato individuato, ma non si tratta di una scacchiera. La funzione *find_redobject()* viene quindi richiamata per capire quali azioni intraprendere. Le operazioni che svolge sono le seguenti:

1. Passaggio della rappresentazione dei colori da RGB ad **HSV** (Hue, Saturation, Value): con HSV è possibile ottenere informazioni migliori sul colore. Si consideri ad esempio la foto di un semaforo rosso. Nella rappresentazione RGB il colore al'interno non è *puro*, ma viene influenzato dalla luce e dalle ombre sulla scena. Con HSV si ha la separazione del colore dalle altre componenti, dunque il valore **hue** non cambia di molto anche in presenza di significative variazioni di luminosità.

2. Applicazione delle trasformazioni morfologiche di erosione e dilatazione: utilizzate per la riduzione del rumore e l'eliminazione di piccoli buchi nel frame.

3. Ricerca di un oggetto che abbia un colore compreso nel range che va dal rosso al giallo.

Nel caso in cui la ricerca non abbia ottenuto alcun risultato, si invia il solito messaggio "clean" nella coda degli ostacoli, altrimenti seguiranno due nuove strade: se il nodo Fog è connesso alla rete Internet, viene chiamata la funzione *send_to_cloud()*; se la connettività invece non è presente, i messaggi *"divert"* e *"no_cloud"* vengono semplicemente spediti tramite RabbitMQ.

***send_to_cloud()*** E' stato individuato un ostacolo, non si tratta di un confine del percorso ed il suo colore è proprio quello che serve per far scattare il sistema di *obstacle_avoidance*. In questo caso, se il nodo è connesso, entra in gioco la funzione *send_to_cloud()*:

1. Ridimensionamento del frame originale

2. Inserimento del frame ridimensionato all'interno del body di un messaggio HTTP usando il metodo POST

3. Attesa della risposta del cloud

4. Invio del messaggio adatto sulla coda *obstacle*

La risposta dell'advanced video recognition (che gira sul Cloud) non può essere attesa a lungo, per questo motivo è presente un timer che impone un tempo massimo di 0.2sec di attesa. Il tempo di elaborazione deve essere infatti inferiore a 382ms e, grazie a questa condizione, è possibile rientrare sempre entro i limiti, evitando così che il rover colpisca qualche ostacolo (questa parte verrà analizzata ed approfondita

nel Capitolo 7). Allo scadere del timer, si ricade nello stesso caso precedente, quando la connettività nel nodo era assente.

Il messaggio "cloud" viene invece spedito quando si riceve, in un tempo utile, una risposta dal sistema di riconoscimento avanzato. In questo caso le risposte possono essere solo di due tipi, "warning" e "divert", e vengono direttamente prelevate dal messaggio HTTP e copiate sulla coda opportuna.

Tutto questo procedimento viene ripetuto in loop fino a quando il tablet risulta essere connesso alla stessa rete del nodo Fog e l'applicazione è attiva, altrimenti si ritorna alla fase iniziale in attesa di una nuova connessione.

## 6.4 Applicazione per tablet Android

L'applicazione è stata realizzata per Android 6.0 e superiori e, durante i vari test e dimostrazioni, è stata installata su un tablet HTC Nexus 9. Il tablet è dotato di antenna GPS integrata + GLONASS, accelerometro, magnetrometro, giroscopio, 2GB di RAM, Wi-Fi 802.11 ac, processore a 64 bit da 2.3 GHz e fotocamera principale da 8MP con autofocus. L'applicazione ha una sola finestra principale, la quale mostra ciò che la fotocamera posteriore sta inquadrando in quell'istante e le varie icone di feedback. Prima dell'installazione su un device, richiede di accedere a fotocamera, bussola, localizzazione, WiFi e blocco schermo; tutti elementi necessari al fine del funzionamento della stessa. All'avvio, dopo la fase di inizializzazione, procede rapidamente alla connessione con RabbitMQ (presente sul Fog) e alla sottoscrizione delle code *compass* ed *obstacle*.

La prima coda viene popolata dai dati ricevuti dalla bussola presente sul tablet. La MainActivity, infatti, implementa l'interfaccia *SensorEventListener*: ogni qual volta c'è un nuovo evento, viene chiamata la funzione *OnSensorChanged()* che riceve il nuovo valore dalla bussola e lo confronta con il precedente. Se i valori sono differenti, allora il nuovo angolo viene pubblicato su *compass*. Questi valori vengono poi prelevati dall'*autonomous driving service* che li sfrutta per l'orientamento spaziale del rover.

Dalla seconda coda, *obstacle*, l'applicazione riceve tutti i messaggi che riguardano la scoperta di ostacoli e l'assenza, o presenza, di connettività verso il cloud. Tali messaggi vengono gestiti in maniera tale da far apparire le relative icone nella schermata principale dell'applicazione ed hanno come unico scopo quello di fornire un feedback visivo.

### 6.4.1 Video streaming – HTTP e MJPEG

Le immagini che vengono catturate dalla fotocamera posteriore devono essere immediatamente trasmesse al *basic video recognition* per essere elaborate. Per tale motivo l'applicazione funziona da server HTTP trasmettendo in streaming il flusso

video sulla porta 8080. La prima versione utilizzava il protocollo RTP (Real Time Protocol - RFC 3550) che risulta più efficiente rispetto a quello HTTP. Fin dalle prime fasi dello sviluppo ci sono stati problemi con OpenCV, il quale non permetteva l'elaborazione delle immagini. Il server RTP trasmetteva infatti il flusso video utilizzando il protocollo UDP[1], mentre OpenCV 3.2 ha degli assert interni al codice che accettano soltanto il flusso TCP se si utilizza RTP. Per questo motivo si è scelto in seguito di passare al protocollo HTTP.

Esistono diverse applicazioni per Android che permettono lo streaming di immagini in tempo reale, ma queste hanno una latenza molto elevata per questo tipo di contesto (circa 300ms). Generalmente quello che si fa è sfruttare la classe pubblica *MediaRecorder*, fornita dalle API Android, che permette di catturare audio e video [28].

```
1  MediaRecorder recorder = new MediaRecorder();
2
3  recorder.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
4  recorder.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
5  recorder.setVideoEncoder(MediaRecorder.VideoEncoder.H263);
6  recorder.setOutputFile(outputFile);
7  recorder.setPreviewDisplay(holder.getSurface());
8
9  recorder.prepare();
10 recorder.start();
11
12 // ...
13
14 recorder.stop();
```

Il video è però accessibile solo dopo aver richiamato la funzione di *stop()*, e questo ovviamente introduce latenza. Per accedere immediatamente al video è possibile altresì utilizzare la `LocalSocket` invece di `File`. Nonostante ciò, quello a cui si accede è un file e non uno stream. Nel caso di MPEG-4 Part 10 (h.264), questo file contiene un'insieme di tantissime informazioni accessorie non utili in questo caso. In aggiunta solo i frame di tipo I (intra) possono essere direttamente decodificati, mentre quelli di tipo P (predictive) hanno necessariamente bisogno dei frame precedenti. L'idea allora è quella di abbandonare la codifica h.264, seppur ottima dal punto di vista del rapporto di compressione, e seguire un'altra strada:

```
1  Camera.PreviewCallback previewCallback = new
       Camera.PreviewCallback()
```

[1]Android non permette di utilizzare RTP con il protocollo TCP

```
2 {
3     public void onPreviewFrame(byte[] data, Camera
          camera)
4     {
5         // Create JPEG
6         YuvImage image = new YuvImage(data, format,
              width, height,
7         null /* strides */);
8         image.compressToJpeg(crop, quality,
              outputStream);
9
10        // Send it over the network ...
11    }
12 };
13
14 camera.setPreviewCallback(previewCallback);
```

Sfruttando l'interfaccia `Camera.PreviewCallback()` è possibile catturare immediatamente l'immagine raw dalla camera, comprimerla in JPEG e trasmetterla tramite il server HTTP. In questo modo i frame che vengono trasmessi sono indipendenti tra loro, potendo così risparmiare banda in caso di congestione e trasmettere un numero inferiori di immagini, riducendo dinamicamente il framerate. Questo approccio è quello seguito dal protocollo MJPEG (Motion JPEG [29]) e l'implementazione utilizzata si rifà al progetto "peepers" presente su GitHub [30].

### 6.4.2 Interfaccia grafica

La Figura 6.3 è uno screenshot scattato sul tablet stesso. Sullo sfondo appare l'immagine di quello che la fotocamera sta inquadrando in questo momento, mentre in basso è possibile trovare delle icone. L'icona di colore rosso col punto esclamativo si attiva nel momento in cui viene rilevata la fine del percorso oppure un ostacolo da evitare assolutamente, quella gialla in caso di ostacoli minori (solo quando c'è connettività con il Cloud), infine quella verde se non sono presenti ostacoli e il rover può proseguire dritto con la marcia. L'icona in alto a destra indica se il rilevamento degli ostacoli è avvenuto sul Cloud o in locale: in caso di assenza di connettività o socket timeout, l'icona sparisce. In basso al centro è presente un pulsante di "RESTART". Alla pressione di questo tasto viene scritto sulla coda *compass* il messaggio "restart". L'autonomous driving service, quando riceve questa istruzione, termina l'esecuzione delle istruzioni per il rover, invia il comando di "idle" e riavvia il server.

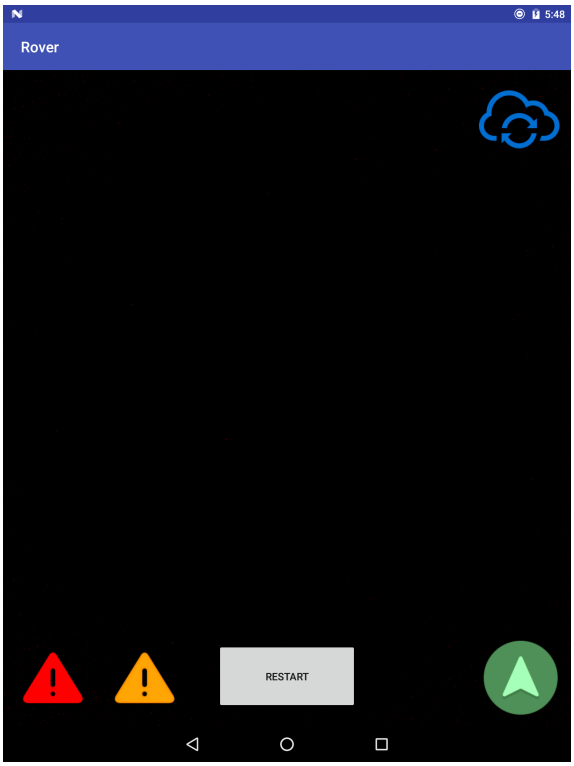



Figura 6.3. Screenshot dell'applicazione in esecuzione sul tablet HTC Nexus 9.

## 6.5 Advanced video recognition

L'***advanced video recognition*** è un container docker che contiene al proprio interno tutte le dipendenze del *basic_video_recognition* (elencate nella sottosezione 6.3.3) ed un nuovo script Python per l'elaborazione video. Il deployment del container è stato effettuato sulla piattaforma Cloud che prende il nome di Sloppy.io e che mette a disposizione 24GB di storage, 1GB di RAM e fino a 50 container (max 5 istanze per singolo container) in esecuzione. All'advanced video recognition sono stati attualmente assegnati 512MB di RAM, è raggiungibile attraverso un indirizzo IP pubblico e risponde ai metodi GET e POST. L'HTTP GET ritorna banalmente una pagina html bianca con il titolo "hi!" che viene scaricata periodicamente dal basic video recognition per stabilire se il Cloud è ancora attivo o meno. Il metodo POST è quello fondamentale per l'elaborazione video: come mostrato in Figura 6.4, il client, che in questo caso è il basic video recognition, invia un'immagine di dimensioni 320x240 compressa in JPEG al'interno del body col metodo HTTP POST; il server esamina questa immagine, controlla quali tipi di ostacoli sono presenti e risponde a sua volta con un messaggio HTTP. Le risposte possono essere di 3 tipi:

1. "clean" – nessun ostacolo trovato

2. “divert” – ostacolo da evitare

3. “warning” – è possibile procedere nonostante via sia un ostacolo

Nella pratica, solo i messaggi “divert” e “warning” vengono inviati al basic video recognition. Come spiegato dettagliatamente nella sottosezione 6.3.3, il basic video recognition invia i frame al Cloud solo quando è strettamente necessario: se nessun tipo di ostacolo è stato trovato con l'algoritmo locale, non ha senso inviare sul Cloud i frame; questo permette di ridurre il traffico scambiato tramite la rete Internet. Tuttavia, ai fini di testing, si è preferito lasciare quel messaggio all'interno del codice dello script, ma durante l'esecuzione non verrà mai utilizzato.

Così come il *basic video recognition*, anche l'*advanced video recognition* presenta al suo interno la funzione *find_redobject()*: dopo aver effettuato la conversione in HSV ed applicato le trasformazioni morfologiche, si occupa di cercare nella scena un oggetto che abbia il colore rosso-arancio. Nel caso in cui la ricerca non abbia portato alcun risultato, l'immagine convertita in HSV viene inviata alla funzione *find_yellowobject()*. Le due funzioni sono praticamente uguali, quello che cambia sono i valori all'interno degli array utilizzati nel processo di mascheramento dei colori. La *find_redobject()*, a differenza di quella utilizzata nel *basic video recognition*, utilizza una maschera basata sul range cromatico HSV che va da 0° a 22°, mentre quella della *find_yellowobject()* va da 22° fino a 50°. Utilizzando questi valori è quindi possibile separare l'identificazione degli oggetti rosso-arancio con quelli di colore giallo.

La scelta dei colori è puramente implementativa ed è stata effettuata al solo scopo di differenziare l'algoritmo locale da quello Cloud. Esistono infatti diversi algoritmi basati sul machine learning, quali K-Means e K-nearest-neighbor, pienamente supportati da OpenCV e che permettono di identificare determinati oggetti presenti sulla scena. L'utilizzo di questi algoritmi però aumenta di gran lunga i tempi di risposta da parte del Cloud e richiede dei server con una potenza di calcolo decisamente maggiore per rientrare all'interno di un intervallo di tempo ragionevole. Sfruttando i colori invece è possibile ottenere tempi estremamente ridotti utilizzando anche servizi di prova come quello fornito da Sloppy.

Richiesta:

```
POST / HTTP/1.1
Host: localhost:8000
Connection: keep-alive
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: python-requests/2.18.4
Content-Length: 10047

......JFIF.............C...........                    .
..
.................. $.' ",#..(7),01444.'9=82<.342...C.

.2!.!2222222222222222222222222222222222222222222222222222.........@..
.....................}.........!1A..Qa."q.2....#B...R..$3br.
.....
%&'()*456789:CDEFGHIJSTUVWXYZcdefghijstuvwxyz.....................
.....................................
.....................w.......!1..AQ.aq."2...B....   #3R..br.
.$4.
%.....&'()*56789:CDEFGHIJSTUVWXYZcdefghijstuvwxyz.................
.......................?..\Pj..Mo...VsE.....K....rU>....#
```

immagine JPEG

Risposta:

```
HTTP/1.0 200 OK
Server: BaseHTTP/0.3 Python/2.7.10
Date: Mon, 18 Sep 2017 07:45:42 GMT
Content-type: text/html

detect
```

Figura 6.4. Esempio di messaggi scambiati tra basic e advanced video recognition.

# Capitolo 7

# Test e validazione

Sono stati condotti una lunga serie di test per verificare il corretto funzionamento del sistema di guida autonoma ed analizzare i tempi di elaborazione necessari per il riconoscimento degli ostacoli. Il risultato di queste prove serve a mostrare il superamento dei requisiti richiesti (evitare l'impatto con gli oggetti e mantenere il rover entro certi confini). Il tempo di elaborazione, l'arrivo dei frame video e la latenza del Cloud sono gli elementi chiave per determinare l'efficienza del prototipo realizzato.

## 7.1 Setup

Il tablet Android comunica con il nodo Fog sfruttando l'access point WiFi collegato sulla porta S0 dello switch frontale. La velocità massima che i due possono instaurare è di 130Mbps, a differenza del rover che può arrivare fino a 72Mbps, a causa della tecnologia della scheda di rete installata. Il tablet, posizionato sopra una staffa a bordo del rover, è rivolto con la fotocamera posteriore verso il basso ed è in grado di inquadrare oggetti fino ad una distanza di circa 20cm oltre le ruote (Figura 7.1).

Di seguito sono elencati i dispositivi e le versioni software utilizzate per i test.

**fogNode:** Nebbiolo fogNode$^{TM}$ NFN 300 con modulo NFL-1000-C
SO: fogOS 1.1.0 beta 3
RAM: 8GB
Storage: SSD SATA-3 120GB
CPU: Core-i5 4402E
Network 1: Gigabit Ethernet IEEE 802.3 ab
Network 2: access point WiFi IEEE 801.11 b/g/n (fino a 130Mbps)

**Tablet:** HTC Nexus 9
SO: Android 7.1.1 (N4F26T)

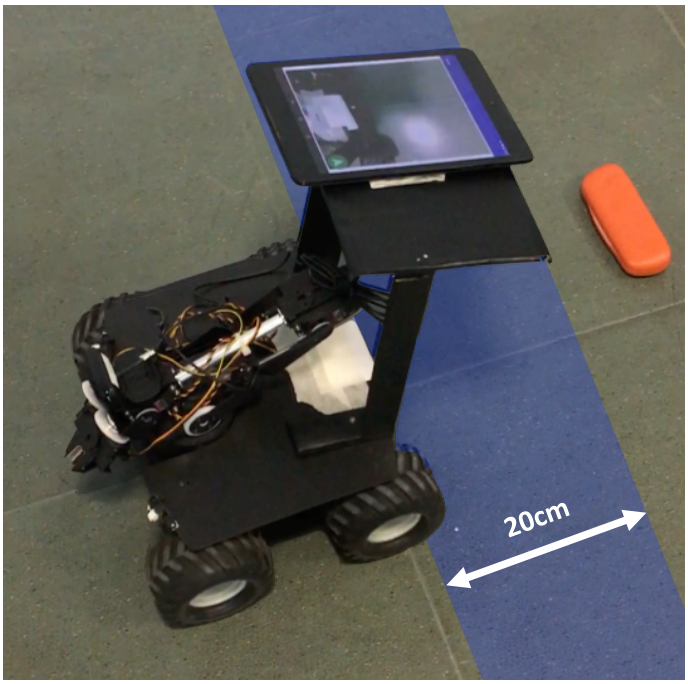



Figura 7.1. Immagine del rover con il tablet Android posizionato nella parte superiore.

CPU: NVIDIA Tegra K1 Dual Denver @ 2.3 GHz a 64 bit
RAM: 2GB
Sensori: Accelerometro - Gyro Sensor - Magnetometro
Fotocamera principale: 8 MP, autofocus, sensore BSI, apertura f/2.4
GPS: Antenna GPS integrata + GLONASS
Wi-Fi: IEEE 802.11 ac, 2x2 MIMO

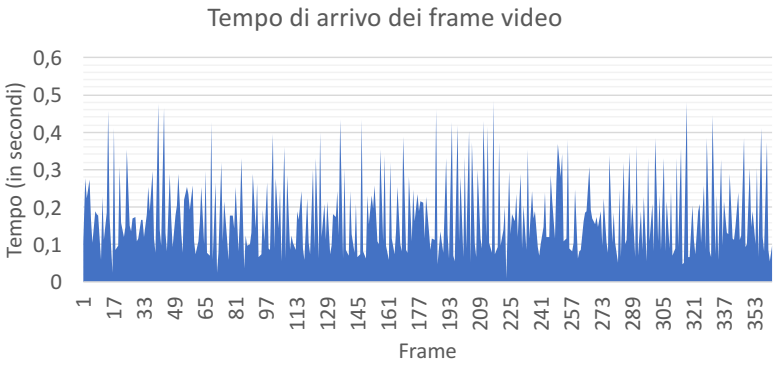
**Rover:** LynxMotion A4WD1
Motori: 4x 12vdc 30:1 200rpm
Sensori: 3x Sharp GP2D12 IR Sensor
Controller: BotBoarduino
Network: Arduino Wi-Fi 101 Shield IEEE 802.11 b/g/n (fino a 72Mbps)
Velocità: massima (da specifiche): 91,44cm/s – a pieno carico: circa 65cm/s

**Container** (disponibili su hub.docker.com)
Message broker: RabbitMQ 3.6 (`rabbitmq:latest`)
Riconoscimento ostacoli (Cloud) : `andreaalfo/advanced_video_recognition:2.0`
Riconoscimento ostacoli (Fog): `andreaalfo/basic_video_recognition:5.2`
Sistema di guida: `andreaalfo/autonomous_driving:3.4`

## 7.2 Test locale

In questo test tutto l'onere dell'elaborazione video è a carico del nodo Fog. L'applicazione sul Cloud è stata temporaneamente fermata ed il sistema fa utilizzo del solo algoritmo locale.

Numero di frame catturati: 362 (Figura 7.2).




| Statistiche | |
|---|---|
| | |
| Media | 0,176329879 |
| Errore standard | 0,005627895 |
| Mediana | 0,138616443 |
| Deviazione standard | 0,107078007 |
| Minimo | 0,008677006 |
| Massimo | 0,483803988 |
| Conteggio | 362 |
| Varianza campionaria | 0,0114657 |

Figura 7.2. Tempo di arrivo dei frame video.

Il tempo minimo registrato per la ricezione di un fotogramma è di soli 8.7ms, mentre quello massimo è circa 483ms. La differenza così elevata tra i due valori è dovuta al ritardo della regolazione automatica di luminosità della camera, agli errori sulla rete e all'algoritmo di codifica (sezione 7.3). In media i frame arrivano ogni 180ms.

Considerata la velocità del rover di circa 23cm/s (utilizzando i parametri settati nel container `autonomous_driving:3.4`) e la distanza minima prima che la camera inquadri un oggetto (circa 20cm), si ottiene il tempo di **impatto** contro un ostacolo pari a 870ms. Tenendo conto inoltre della latenza dovuta all'invio del messaggio di *stop* al rover e all'effettivo bloccaggio delle ruote; il tempo a disposizione scende di altri 5ms[1], riducendosi a 865ms.

Affinchè il sistema riesca ad evitare un ostacolo occorre che:

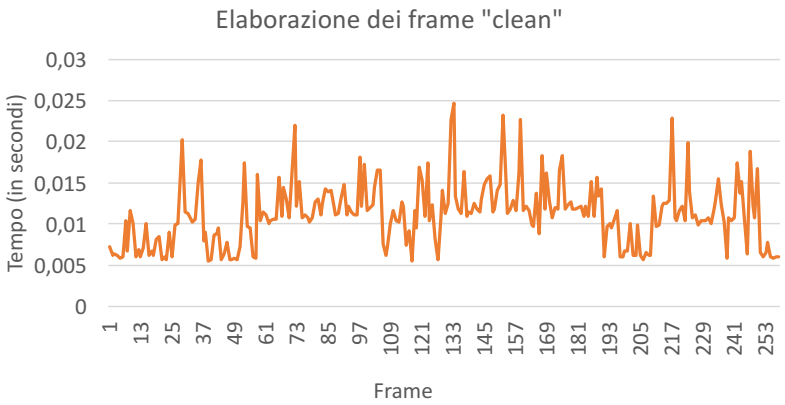
$$\boxed{t_{\text{elaborazione\_video}} + t_{\text{ricezione\_frame}} < t_{\text{impatto}}}$$

Nel caso peggiore il $t_{\text{ricezione\_frame}}$ è pari a 483ms ed il $t_{\text{impatto}}$ è di 865ms, dunque il $t_{\text{elaborazione\_video}}$ deve essere inferiore a 865ms - 483ms = 382ms.

L'analisi dei 362 frame ha prodotto i seguenti risultati:

[1]Si tratta di una stima, non è possibile misurare sperimentalmente questo valore.

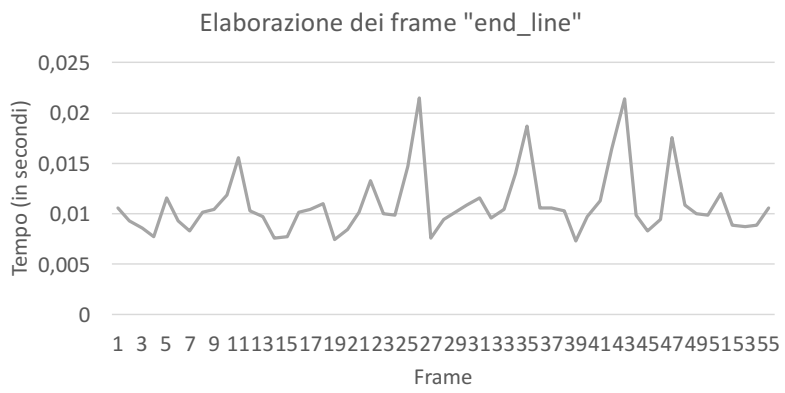
- 258 non contengono ostacoli (*clean*)
- 55 contengono una scacchiera che delimita la fine del percorso (*end_line*)
- 49 contengono un ostacolo di colore compreso tra rosso e giallo (*divert*)




| Statistiche | |
|---|---|
| | |
| Media | 0,011181063 |
| Errore standard | 0,000234515 |
| Mediana | 0,011100411 |
| Deviazione standard | 0,003766864 |
| Minimo | 0,005518913 |
| Massimo | 0,024699926 |
| Conteggio | 258 |
| Varianza campionaria | 1,41893E-05 |

Figura 7.3. Elaborazione dei frame *clean*.

I tempi necessari per l'elaborazione dei frame di tipo *clean* sono riportati in Figura 7.3. Il valore minimo è di soli 5ms, quello massimo di 24ms. Con questi frame siamo ben al di sotto del limite dei 382ms calcolato precedentemente.



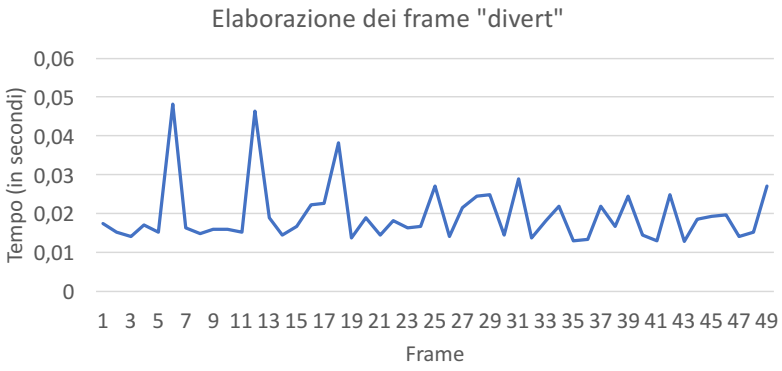

| Statistiche | |
|---|---|
| | |
| Media | 0,010899184 |
| Errore standard | 0,00042622 |
| Mediana | 0,010140896 |
| Deviazione standard | 0,003160936 |
| Minimo | 0,007286787 |
| Massimo | 0,021463871 |
| Conteggio | 55 |
| Varianza campionaria | 9,99152E-06 |

Figura 7.4. Elaborazione dei frame *end_line*.

Analogamente, anche per i frame di tipo *end_line* si registrano bassi tempi di elaborazione, addirittura inferiori rispetto ai primi. Nel caso peggiore, infatti, il

*basic_video_recognition* ha impiegato poco meno di 22ms prima di individuare la scacchiera posta ai confini del percorso.

In presenza di ostacoli, i tempi risultano essere leggermente superiori ai precedenti, ma tuttavia abbondamente sotto la soglia limite (Figura 7.5).




| Statistiche | |
|---|---|
| | |
| Media | 0,019704269 |
| Errore standard | 0,001100192 |
| Mediana | 0,016812086 |
| Deviazione standard | 0,007701342 |
| Minimo | 0,012892008 |
| Massimo | 0,048266888 |
| Conteggio | 49 |
| Varianza campionaria | 5,93107E-05 |

Figura 7.5. Elaborazione dei frame *divert*.

Per concludere, facendo un confronto tra i vari campioni misurati (Figura 7.6), è possibile affermare che, indipendentemente dal tipo di frame, l'elaborazione avviene in tempi estremamente ridotti. I frame che contengono ostacoli di colore rosso-giallo sono quelli per cui è necessario sfruttare più a lungo la CPU del nodo Fog, ciononostante non si registrano mai valori oltre i 50ms.

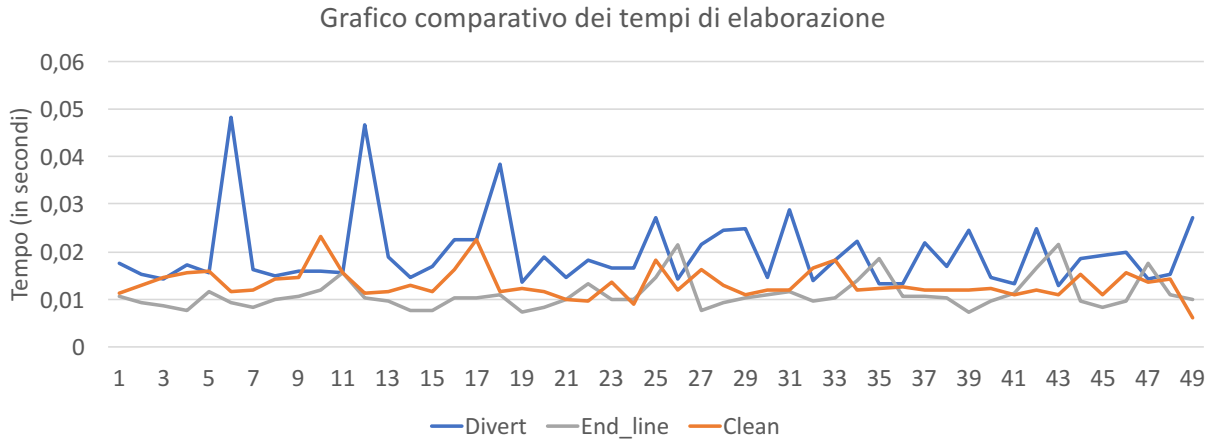
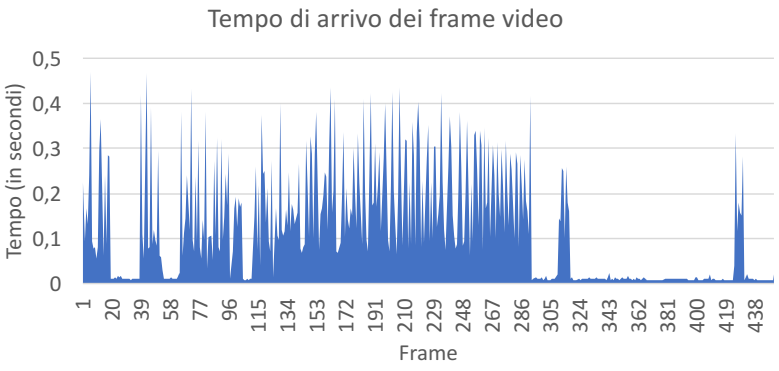



Figura 7.6. Confronto tra i tempi di elaborazione dei diversi frame.

## 7.3 Test Cloud

Nel seguente test la connettività verso il Cloud è stata ripristinata e l'applicazione locale collabora attivamente con l'*advanced_video_recognition* per effettuare le opportune scelte di guida.

Numero di frame catturati: 450 (Figura 7.7).




| Statistiche | |
|---|---|
| | |
| Media | 0,120028682 |
| Errore standard | 0,005781191 |
| Mediana | 0,084645629 |
| Deviazione standard | 0,122637579 |
| Minimo | 0,007055044 |
| Massimo | 0,468852043 |
| Conteggio | 450 |
| Varianza campionaria | 0,015039976 |

Figura 7.7. Tempo di arrivo dei frame video.

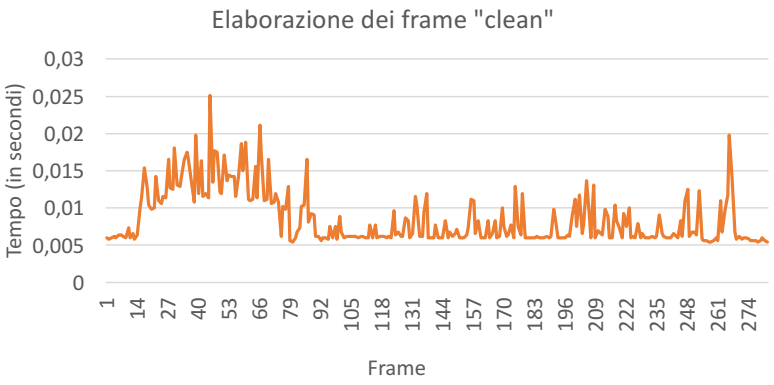
I dati ricavati da questo grafico sono in linea con quelli del test eseguito sfruttando solo l'algoritmo locale. Il tempo di arrivo dei frame non viene infatti influenzato dalla presenza o dall'assenza di connettività. E' possibile però vedere delle *anomalie* in alcuni intervalli. In particolare, tra il frame n.320 e il frame n.421, i tempi di arrivo sono decisamente molto bassi. In teoria questa situazione è quella "tradizionale": in questo test, ed in maniera molto più accentuata in quello precedente, il tablet veniva mosso continuamente in diverse posizioni della stanza, inviando immagini molto differenti tra loro. Questo comporta maggiori ritardi per la cattura da parte della fotocamera (a causa delle continue modifiche della luminosità) e soprattutto una perdita significativa delle prestazioni dell'algoritmo di codifica. Nel caso in cui l'immagine inquadrata è molto uniforme (ad esempio il pavimento di una stanza o nel caso pratico un terreno agricolo), l'algoritmo di compressione MJPEG (sottosezione 6.4.1) riesce a ridurre le dimensioni dei singoli frame fino a 20 volte rispetto agli originali non compressi, lavorando anche molto più velocemente. Il caso tipico prevede quindi la presenza di pochi picchi, qui invece si è voluto testare il sistema in situazioni più "estreme" per sottolineare il suo funzionamento anche nelle condizioni peggiori.

L'analisi dei 450 frame ha rilevato che:

- 282 non contengono ostacoli (*clean*)

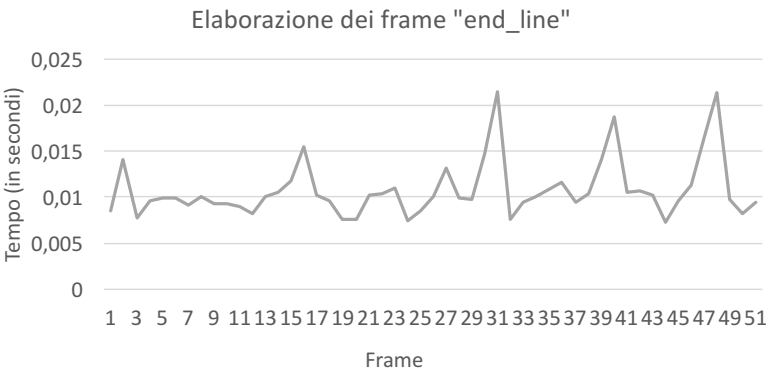
- 51 contengono una scacchiera che delimita la fine del percorso (*end_line*)
- 53 contengono un ostacolo di colore rosso-arancio (*divert*)
- 64 contengono un ostacolo di colore giallo (*warning*)

I tempi necessari per l'elaborazione dei frame di tipo *clean* ed *end_line* non presentano differenze con quelli analizzati nel test precedente (Figure 7.8 e 7.9).




| Statistiche | |
|---|---|
| | |
| Media | 0,008604373 |
| Errore standard | 0,000214575 |
| Mediana | 0,006566048 |
| Deviazione standard | 0,003603321 |
| Minimo | 0,005424976 |
| Massimo | 0,025092125 |
| Conteggio | 282 |
| Varianza campionaria | 1,29839E-05 |

Figura 7.8. Elaborazione dei frame *clean* (Cloud).



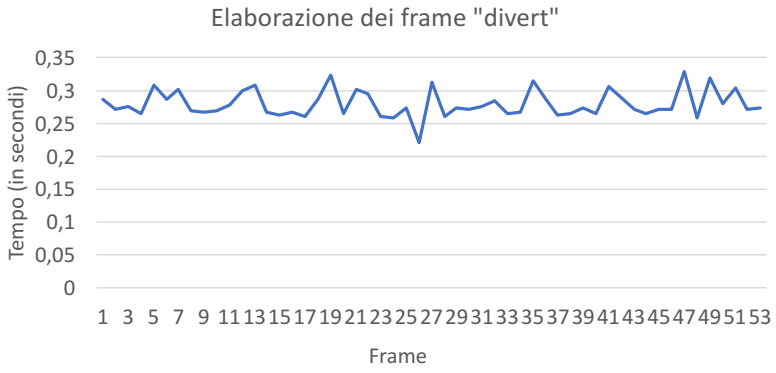

| Statistiche | |
|---|---|
| | |
| Media | 0,010818126 |
| Errore standard | 0,000443017 |
| Mediana | 0,010087967 |
| Deviazione standard | 0,003163774 |
| Minimo | 0,007286787 |
| Massimo | 0,021463871 |
| Conteggio | 51 |
| Varianza campionaria | 1,00095E-05 |

Figura 7.9. Elaborazione dei frame *end_line* (Cloud).

La loro elaborazione avviene infatti solo in locale: l'assenza di ostacoli evita l'invio di questi frame sul Cloud, ottenendo un risparmio in termini di latenza e banda utilizzata.

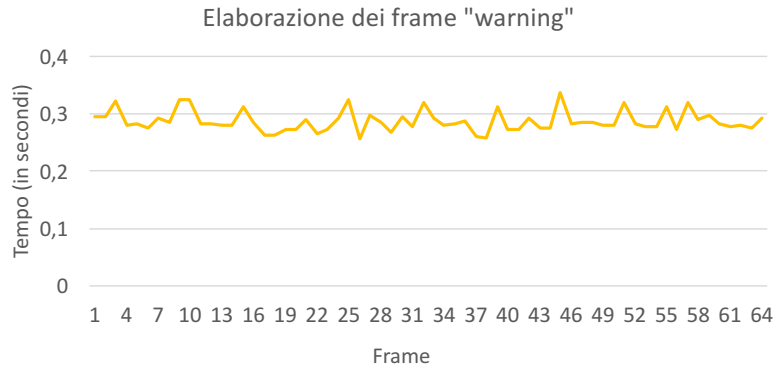
La situazione risulta essere differente per i frame che presentano al loro interno ostacoli che hanno un colore compreso tra il rosso ed il giallo. In questo caso il

sistema si occupa di inviare le immagini al Cloud ed attende, per un tempo massimo di 200ms, la risposta HTTP contenente la tipologia dell'ostacolo. L'invio e l'attesa sono operazioni onerose dal punto di vista della tempistica, che in questo caso subisce dei forti incrementi.




| Statistiche | |
|---|---|
| | |
| Media | 0,279182919 |
| Errore standard | 0,002806374 |
| Mediana | 0,272849083 |
| Deviazione standard | 0,003603321 |
| Minimo | 0,220523834 |
| Massimo | 0,328447104 |
| Conteggio | 53 |
| Varianza campionaria | 0,000417414 |

Figura 7.10. Elaborazione dei frame *divert* (Cloud).




| Statistiche | |
|---|---|
| | |
| Media | 0,287053339 |
| Errore standard | 0,002264703 |
| Mediana | 0,282379985 |
| Deviazione standard | 0,018117625 |
| Minimo | 0,256342888 |
| Massimo | 0,336853981 |
| Conteggio | 64 |
| Varianza campionaria | 0,000328248 |

Figura 7.11. Elaborazione dei frame *warning* (Cloud).

Per i frame *divert* l'elaborazione totale (invio del frame, attesa risposta, scrittura ed invio messaggio sul server RabbitMQ) richiede dai 220ms fino ai 320ms (Figura 7.10), per quelli *warning* (Figura 7.11) raggiunge dei picchi di oltre 330ms.

Mettendo a confronto i vari grafici, si nota chiaramente (Figura 7.12) che i frame elaborati in locale impiegano tempi significativamente minori rispetto a quelli che vengono trasmessi all'*advanced_video_recognition*. Per questo motivo si preferisce utilizzare l'elaborazione remota solo quando necessario.

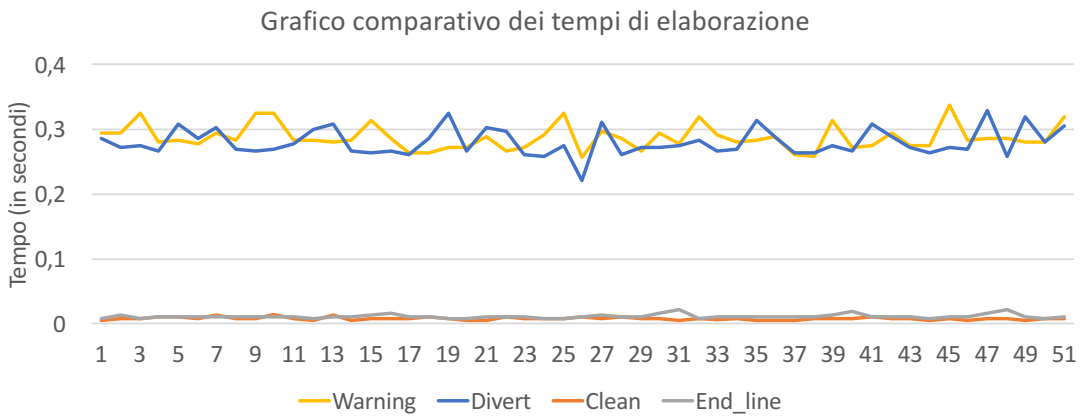



Figura 7.12. Confronto tra i tempi di elaborazione dei diversi frame (Fog e Cloud).

Nonostante il Cloud abbia ridotto, in determinate circostanze, la risposta del sistema di guida, questa risulta essere ancora valida. La situazione *limite* analizzata nel primo test, infatti, richiedeva un tempo di elaborazione inferiore ai 382ms. Durante le varie prove non sono mai stati registrati valori che superassero i 340ms: la soglia dei 200ms permette di scartare le risposte ricevute dal sistema di riconoscimento avanzato se questo impiega troppo tempo (congestione, CPU sovraccarica, ecc..), riuscendo a restare **sempre** sotto i limiti consentiti.

Infine, in Figura 7.13, è presente un confronto dei tempi di elaborazione del sistema mostrato nel test precedente (solo locale) e quello analizzato in questa sezione.

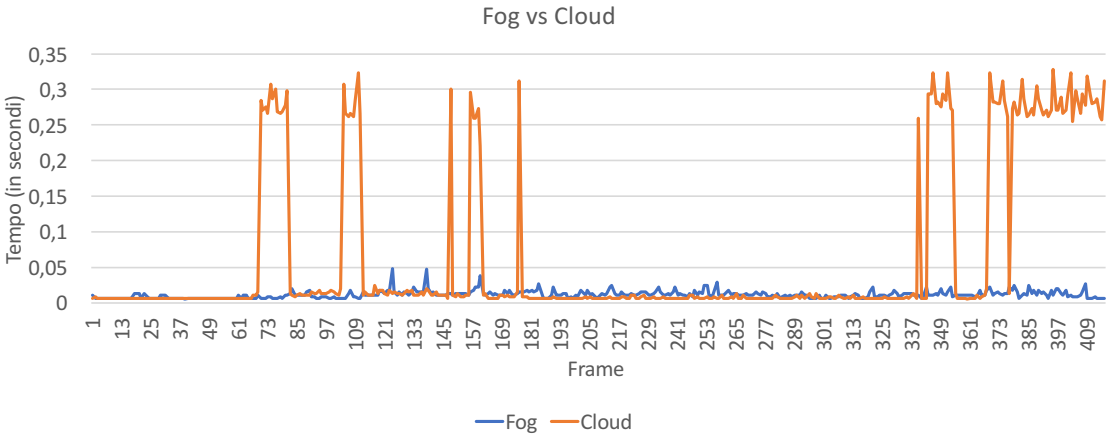



Figura 7.13. Confronto tra Fog e Cloud.

# Capitolo 8

# Conclusioni e lavori futuri

In questa tesi è stato sviluppato il prototipo di un sistema di guida autonoma in grado di comandare i movimenti di un rover, ai fini di seguire un percorso prestabilito ed evitare gli ostacoli lungo il suo cammino. Il software delegato a questo scopo è composto da varie parti che scambiano tra loro messaggi ed informazioni sfruttando protocolli standard quali HTTP e AMQP.

In questo capitolo conclusivo sono stati riassunti i punti critici di questo sistema e quali miglioramenti potranno essere fatti in futuro.

## 8.1 Controllo a catena aperta

Uno dei fattori che ha causato maggiori problemi, durante le fasi dello sviluppo di questo sistema, è stato il controllo del rover. Come accennato nei precedenti capitoli, l'A4WD1 non dispone di sensori che possano rilevare la velocità istantanea, lo stato della batteria o il bloccaggio di una o più ruote. Per questa ragione è stato difficile riuscire a fare effettuare le varie manovre al rover in maniera corretta.

L'utilizzo della bussola del tablet Android ha permesso di ottenere un'informazione, poco precisa, sulla direzione percorsa. In particolare, questa soluzione risulta essere troppo influenzabile da fattori esterni. Avvicinando infatti un comune smartphone al tablet, si otterrà la lettura di valori errati e di conseguenza il sistema guida intraprenderà delle scelte non corrette durante la fase di rotazione.

Il controllo del rover è quindi definibile a *"catena aperta"*: la riduzione o l'aumento della velocità dovuta al consumo della batteria, un blocco delle ruote causato da un fattore esterno o il semplice cambio di pavimentazione non possono essere percepiti dall'applicazione che lo controlla.

## 8.2 Possibili miglioramenti

Uno dei miglioramenti che potrebbe essere fatto in futuro è quello di installare dei sensori a bordo dell'A4WD1, oppure utilizzare un modello differente che sia già previsto di sistemi odometrici. In questo caso, la precisione e l'affidabilità del sistema di guida ne gioverebbero incredibilmente. Con la presenza di informazioni aggiuntive, è possibile inoltre utilizzare anche degli algoritmi di guida autonoma e path finder più complessi, mediante l'utilizzo del framework **ROS** - *Robot Operating System* - che fornisce di base tool e strumenti utili per la realizzazione di applicazioni per il mondo della robotica.

L'unico mezzo di visione di cui dispone questo sistema è la fotocamera posteriore del tablet Android, che invia i frame video attraverso la rete WiFi. L'utilizzo di un singolo dispositivo di acquisizione restituisce delle immagini monodimensionali; la difficoltà di riconoscere correttamente gli ostacoli risulta essere molto elevata e la visione può presentare alcuni angoli ciechi. Utilizzare più camere, abbinate anche a sensori di profondità (così come accade con il Microsoft Kinect), permette di implementare un miglior sistema di riconoscimento ostacoli.

L'utilizzo del servizio di base di Cloud proposto da Sloppy.io fornisce dei server tradizionali che dispongono di una potenza di calcolo pari a quella di un personal computer. Acquistando soluzioni enterprice da Google o Amazon, è possibile caricare sul Cloud un'applicazione di *obstacle detection* basata su algoritmi di machine learning, ottenendo tuttavia delle risposte in tempi brevi.

## 8.3 Implementazione su veicolo

Il passaggio dalla dimostrazione con il rover all'implementazione sul veicolo potrebbe essere effettuato fin da subito. L'utilizzo di un trattore vero e proprio permetterebbe anche di sfruttare il sistema GPS o GNSS per poter stabilire i "confini" e decidere il path planning da seguire. Il nodo Fog che è stato utilizzato non è adatto a questo scenario, poiché ha necessità di essere alimentato a 220v, tuttavia sono già in fase di produzione alcuni nodi più piccoli e con un diverso tipo di alimentazione. Su un trattore potrebbero essere installati, oltre alle camere, sistemi di individuazione degli ostacoli laser lidar e strumenti necessari ai fini del rilevamento di sostanze nel terreno.

La connettività perenne del nodo Fog non è affatto necessaria e quindi è possibile inviare il trattore anche in zone in cui il segnale cellulare o quello WiFi della base operativa non sono presenti. Con la collaborazione che è stata avviata con alcune aziende specializzate in questo campo, magari in futuro sarà possibile vedere la realizzazione e l'implementazione di questo sistema su larga scala.

# Bibliografia


[1] OpenFog Consortium. *Definition of Fog Computing*. URL: https://www.openfogconsortium.org/resources/#definition-of-fog-computing.

[2] OpenFog Consortium. *OpenFog Reference Architecture for Fog Computing*. Feb. 2017. URL: https://www.openfogconsortium.org/wp-content/uploads/OpenFog_Reference_Architecture_2_09_17-FINAL.pdf.

[3] OpenFog Consortium. *Fog Computing use cases*. 2017. URL: https://www.openfogconsortium.org/resources/#use-cases.

[4] OpenFog Consortium. *Out of the Fog: High-Scale Drone Package Delivery*. Giu. 2016. URL: https://www.openfogconsortium.org/wp-content/uploads/OpenFog-Transportation-Drone-Delivery-Use-Case.pdf.

[5] Nebbiolo Technologies. *Bridging the intelligence gap between the cloud and IoT and end-points*. URL: https://www.nebbiolo.tech/platform/.

[6] Nebbiolo Technologies. *fogNode Overview*. 2017. URL: https://www.nebbiolo.tech/wp-content/uploads/fogNode-OVERVIEW-rev3.pdf.

[7] Nebbiolo Technologies. *Nebbiolo NFN-300 Series fogNodeTM - Datasheet Ver. 1.4*. 2016. URL: https://www.nebbiolo.tech/wp-content/uploads/NFN-300-datasheet-v1.4-2.pdf.

[8] OPC Foundation. *OPC Uni ed Architecture*. URL: https://opcfoundation.org/wp-content/uploads/2014/05/OPC-UA_Overview_IT.pdf.

[9] Steer Davies Gleave. *Research for TRAN Committee – Self-piloted cars: The future of road transport?* 2016. URL: https://doi.org/10.2861/66390.

[10] «Convenzione sulla circolazione stradale». In: Vienna, nov. 1968. URL: http://www.meltingpot.org/IMG/pdf/convenzione_vienna_8_novembre_1968-2.pdf.

[11] Transport Regulators Align Control Enforcement. *Spiegazione del regolamento (CE) n. 561/2006*. URL: http://www.euro-controle-route.eu/site/files/tekstfotos/TRACE_IT.pdf.

[12] John Reid et al. *Autonomous Driving in Agriculture Leading to Autonomous Worksite Solutions*. Set. 2016. URL: https://doi.org/10.4271/2016-01-8006.

[13] William J. Hughes Technical Center. *GPS Performance Analysis Report*. Gen. 2017. URL: http://www.nstb.tc.faa.gov/reports/PAN96_0117.pdf.

[14] Mark & Lachapelle Gérard B Ong Richard & Petovello. «Assessment of GPS/-GLONASS RTK under various operational conditions». In: 6 (gen. 2009). URL: https://www.researchgate.net/publication/268357703_Assessment_of_GPSGLONASS_RTK_under_various_operational_conditions.

[15] *Docker website*. URL: https://www.docker.com/what-docker.

[16] FreeBSD. *Capitolo 15. Jail*. URL: https://www.freebsd.org/doc/it_IT.ISO8859-15/books/handbook/jails-intro.html.

[17] Docker Inc. *Docker container networking*. URL: https://docs.docker.com/engine/userguide/networking/.

[18] CloudAMQP. *What is RabbitMQ?* Ago. 2015. URL: https://www.cloudamqp.com/blog/2015-05-18-part1-rabbitmq-for-beginners-what-is-rabbitmq.html.

[19] RabbitMQ. *Finding bottlenecks with RabbitMQ 3.3*. 2014. URL: http://www.rabbitmq.com/blog/2014/04/14/finding-bottlenecks-with-rabbitmq-3-3/.

[20] OpenCV Dev Team. *Basic Structures*. URL: http://docs.opencv.org/2.4/modules/core/doc/basic_structures.html.

[21] *OpenCV website*. URL: http://opencv.org.

[22] LynxMotion. *About the A4WD1 Robot*. URL: http://www.lynxmotion.com/c-119-auton-combo-kit.aspx.

[23] LynxMotion. *About the BotBoarduino*. URL: http://www.lynxmotion.com/c-153-botboarduino.aspx.

[24] LynxMotion. *About the BotBoard II*. URL: http://www.lynxmotion.com/p-252-bot-board-ii.asp.

[25] Arduino. *Arduino Duemilanove*. URL: https://www.arduino.cc/en/Main/ArduinoBoardDuemilanove.

[26] Arduino. *ARDUINO WIFI 101 SHIELD*. URL: https://store.arduino.cc/usa/arduino-wifi-101-shield.

[27] OpenCV Dev Team. *Morphological Transformations*. 2014. URL: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html.

[28] Foxdog Studios. *Peepers: realtime video streaming from Android*. Feb. 2013. URL: https://foxdogstudios.com/peepers.

[29] Library of Congress. *MJPEG (Motion JPEG) Video Codec*. URL: https://www.loc.gov/preservation/digital/formats/fdd/fdd000063.shtml.

[30] Foxdog Studios. *A simple IP camera application for Android*. URL: https://github.com/foxdog-studios/peepers.