



ABSTRACT

I sistemi di posizionamento ferroviari e ferrotramviari ad oggi impiegati, fanno un largo uso di apparati installati a terra e segnali provenienti dalla linea. La loro realizzazione ha pertanto un costo e un impatto ambientale non trascurabili. Per questo motivo, è necessario pianificare una migrazione verso di sistemi di posizionamento autonomi, in accordo alle normative operazionali europee in ambito ferroviario e ferrotramviario definite dallo standard ERTMS/ETCS.

Un sistema di posizionamento ferroviario, o ferrotramviario, è autonomo quando non fa alcun uso di apparati installati a terra.

In questa Tesi si mostrano, e discutono, i risultati sperimentali ottenuti attraverso un' attività di *fault injection* condotta su un sottosistema di posizionamento ferrotramviario autonomo.

Il sistema *target* dell'analisi basa il suo funzionamento sull'utilizzo di un insieme di sensori installati a bordo treno, le cui misure campionate vengono processate da un algoritmo noto come *Sensor Fusion Algorithm* (SFA).

SFA è un algoritmo che integra le misure fornite da un insieme di sensori al fine di attutirne il rumore di misura. L'output prodotto da SFA è una misura più sicura e affidabile di quella che si otterrebbe considerando i sensori singolarmente. In questo contesto, la misura che si intende fornire attraverso l'uso di SFA è la posizione del treno.

Per le sue caratteristiche architettoniche, è possibile classificare il sistema come un *Cyber Physical Systems of Systems* (CPSoS), mentre il particolare dominio applicativo colloca il sistema nell'area *safety-critical*.

La Tesi passa in rassegna lo stato dell'arte circa la valutazione della *dependability* di un sistema e le tradizionali tecniche di posizionamento ferroviario. Segue poi una descrizione del sistema, del suo contesto operativo nominale e degli standard che lo regolamentano. Si descrive l'ambiente in cui il sistema verrà analizzato, e infine si discutono i risultati dell'analisi condotta.

Attraverso l'attività di *fault injection* è stato principalmente osservato che il sistema è in grado di tollerare bene guasti al sistema di comunicazione verso i sensori, a condizione che rimanga funzionante almeno un canale: quello verso il *sensore inerziale*.

Il sistema sembra inoltre capace di individuare, e correggere di conseguenza, i messaggi ricevuti che hanno un' elevata probabilità di contenere un dato sbagliato.

ABSTRACT

Traditional railway positioning subsystems heavily relies upon ground instruments and track circuits signals. Hence their realization has a non negligible cost and environmental impact. For this reason, it is necessary to plan a migration to autonomous positioning subsystems, according to european operational regulations as defined in ERTMS/ETCS standard.

A railway, or tramway, positioning system is called autonomous when no ground instrument is used.

In this Thesis, the results of a fault injection campaign performed against an autonomous tramway positioning subsystem are provided and discussed.

The target system bases its operation on the use of a set of sensors installed on board the train, whose measurements are processed by an algorithm known as Sensor Fusion Algorithm (SFA).

SFA works by integrating the measurements provided by a set of sensors, in order to reduce the measurement noise. SFA output is a more secure and reliable measure than the one that would be obtained considering the sensors individually. In this context, the measure intended to be provided through the use of SFA is the position of the train.

Due to its architectural elements, it is possible to classify the system as a Cyber Physical Systems of Systems (CPSoS), and the particular application domain classifies the system as safety critical.

The Thesis reviews the state-of-the-art regarding the evaluation of the dependability of a system and the traditional techniques of railway positioning.

It continues with a system description, its nominal operating context and the standards that regulate it.

The environment in which the system will be analyzed is described, and finally the results of the analysis conducted are discussed.

Through the activity of fault injection it was mainly observed that the system is able to tolerate faults with respect to the communication system towards the sensors, provided that at least one channel remains functional: the one towards the inertial measurement unit.

The system also seems able to identify, and correct, the messages received that have a high probability to contain wrong data.

INDICE

1	Introduzione	9
1.1	Dependability	10
1.2	Osservazione dei sistemi	13
1.3	Fault Injection	15
1.4	Scopo della Tesi	17
2	Stato dell'arte	19
2.1	Sistemi ferroviari e ferrotramviari	19
2.1.1	La safety nei sistemi ferroviari	20
2.2	Il problema del posizionamento	21
2.2.1	Tradizionali tecniche di posizionamento	21
2.2.2	Verso ETCS-3	23
2.3	Cyber Physical Systems of Systems	25
3	Il sistema analizzato	29
3.1	Descrizione generale	29
3.2	Constituent Systems	30
3.2.1	Sensor Set	30
3.2.2	Piattaforma di elaborazione dati	31
3.2.3	OBCU	32
3.3	Specifiche delle interfacce	33
3.3.1	RUI	33
3.4	Interazioni	34
3.4.1	Acquisizione dei dati	35
3.4.2	Trasmissione della posizione	35
3.4.3	Interazioni con eventuali operatori umani	36
3.5	Architettura Software	37
3.5.1	Sensor Fusion Library	37
3.5.2	Listener	38
3.5.3	Interface Modules	38
4	Ambiente di analisi	41
4.1	Il tool di analisi	41
4.1.1	Load Generator	41
4.1.2	Injector	43
4.1.3	Monitor	44
4.1.4	Controller	45
4.2	Note sull'intrusività	45

5 Attività Sperimentale	47
5.1 Misure di interesse	47
5.2 Guasti iniettati	48
5.3 Esperimenti	48
5.3.1 Golden Run	49
5.3.2 Soppressione della comunicazione odometro - SFA	50
5.4 Considerazioni finali	59
6 Conclusioni	61

ELENCO DELLE TABELLE

Tabella 1	Livelli SIL	21
Tabella 2	Specifiche Tecniche NVidia TX-Jetson	33
Tabella 3	Specifiche delle RUPI del sistema	33
Tabella 4	Specifiche delle RUMI del sistema	35
Tabella 5	Moduli di <i>interface-modules</i>	39
Tabella 6	Golden Run: workload	49
Tabella 7	Golden Run: Risultati	50
Tabella 8	Esperimento 3.1: workload	50
Tabella 9	Esperimento 3.1: Risultati	50
Tabella 10	Esperimento 3.1: Confronto con Golden Run	51
Tabella 11	Esperimento 3.2: workload	53
Tabella 12	Esperimento 3.2: Risultati	53
Tabella 13	Esperimento 3.2: Confronto con golden run	54
Tabella 14	Esperimento 3.3: workload	56
Tabella 15	Esperimento 3.3: Risultati	56
Tabella 16	Esperimento 3.3: Confronto con golden run	57

ELENCO DELLE FIGURE

Figura 1	Fallimento e restauro	9
Figura 2	La <i>safety</i> estende il concetto di <i>reliability</i>	11
Figura 3	Catena guasto errore fallimento	12
Figura 4	Il ciclo di vita dei sistemi: modello a V	13
Figura 5	Monitoring black-box	14
Figura 6	Monitoring white-box	14
Figura 7	Elementi di un processo <i>fault injection</i>	15
Figura 8	Schema di un tipico scenario ferrotramviario	20
Figura 9	Livelli ETCS	23
Figura 10	Schema dell'ambiente di <i>fault injection</i>	24
Figura 11	Interfacce di un CPS	26
Figura 12	RUMI e RUPI	27
Figura 13	Schema SFA	29
Figura 14	<i>Inertial Measurement Unit</i>	31
Figura 15	Ricevitore GPS ublox EVK-M8T	32
Figura 16	Nvidia TX-Jetson	32
Figura 17	Modem TP-LINK M7350 LTE-4G	34
Figura 18	Diagramma a blocchi del sottosistema di posizionamento	34
Figura 19	Sequenza di acquisizione dati	36
Figura 20	Sequenza di trasmissione della posizione	36
Figura 21	Diagramma a blocchi di <i>listener</i>	38
Figura 22	Diagramma a blocchi del sistema software	39
Figura 23	Interfaccia utente verso <i>SynthDataGen</i>	42
Figura 24	Diagramma a blocchi <i>SynthDataGen</i>	43
Figura 25	Interfaccia utente di RTT	44
Figura 26	Controller RTT	45
Figura 27	File di log	45
Figura 28	Report HTML prodotto da RTT	46
Figura 29	Traccia di analisi	47
Figura 30	Esperimento 3.1: Grafico di confronto ECEF X con Golden Run	51
Figura 31	Esperimento 3.1: Grafico di confronto ECEF Y con Golden Run	52

8 Elenco delle figure

- Figura 32 Esperimento 3.1: Grafico di confronto ECEF Z con Golden Run 52
- Figura 33 Esperimento 3.2: Grafico di confronto ECEF X con Golden Run 54
- Figura 34 Esperimento 3.2: Grafico di confronto ECEF Y con Golden Run 55
- Figura 35 Esperimento 3.2: Grafico di confronto ECEF Z con Golden Run 55
- Figura 36 Esperimento 3.3: Grafico di confronto ECEF X con Golden Run 57
- Figura 37 Esperimento 3.3: Grafico di confronto ECEF Y con Golden Run 58
- Figura 38 Esperimento 3.3: Grafico di confronto ECEF Z con Golden Run 58

INTRODUZIONE

Negli ultimi anni, i sistemi informatici hanno assunto un ruolo sempre più centrale nelle attività umane.

Inizialmente, il computer era considerato un semplice strumento di supporto alla matematica applicata, capace di svolgere calcoli particolarmente onerosi in un tempo relativamente breve. Con lo sviluppo delle tecnologie, i sistemi informatici sono ad oggi impiegati in un vasto insieme di domini applicativi, dall'elettromedicale al trasporto aereo, fino all'*Internet of Things*.

Quanto più si diffonde l'utilizzo dei sistemi informatici, tanto più peso assume un eventuale fallimento dei medesimi.

La letteratura scientifica dimostra che la valutazione della *dependability* di un sistema informatico è un problema chiave.

Per *dependability* si intende la capacità che ha un sistema di fornire un servizio in modo corretto. [1]

Un *fallimento* è una transizione compiuta da un sistema dall'erogazione di un servizio corretto verso l'erogazione di un servizio scorretto. La transizione contraria è detta *restauro*.

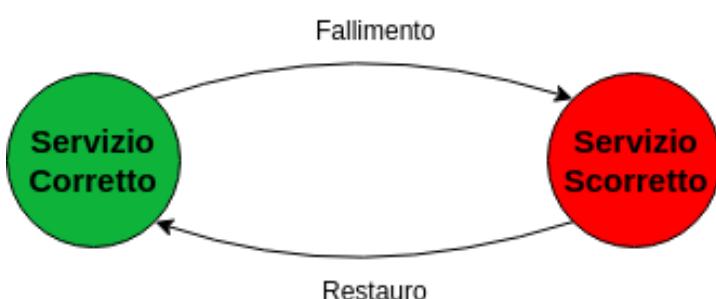


Figura 1: Fallimento e restauro

Effettuare misure sperimentalali su un sistema informatico, o su un suo prototipo, è una valida opzione per valutarne la *dependability*.

1.1 DEPENDABILITY

La *dependability* di un sistema è:

- Misurata rispetto a un certo insieme di proprietà note come *measures*;
- Raggiunta attraverso l'utilizzo di specifiche tecniche, i *means*;
- Minacciata dai *threats*, ossia da tutto ciò che porta il sistema ad erogare un servizio improprio.

Un sistema può fallire nel caso in cui questo non sia conforme alle specifiche, oppure perchè le specifiche non descrivono adeguatamente le sue funzioni.

La *dependability* di un sistema viene misurata rispetto alle seguenti proprietà:

- *Availability*: L'alternanza tra la fornitura di un servizio corretto e uno scorretto.

$$A(t) = \begin{cases} 1 & \text{se il servizio fornito è corretto al tempo } t \\ 0 & \text{altrimenti} \end{cases}$$

$E[A(t)]$: probabilità che il servizio fornito sia corretto al tempo t

- *Reliability*: Capacità di fornire un servizio continuamente corretto in un certo intervallo di tempo.

$R(t)$: probabilità di fornire un servizio corretto nell'intervallo $[0, t]$

- *Safety*: Il tempo medio a un fallimento catastrofico.

$S(t)$: probabilità che non si verifichi alcun fallimento catastrofico nell'intervallo $[0, t]$

- *Time to Failure*: Il tempo che intercorre fra l'ultimo restauro e il successivo fallimento.
Spesso è opportuno considerare il valore atteso di questa grandezza, il *Mean Time to Failure* (MTTF).
- *Maintainability*: Il tempo necessario a restaurare il sistema, dopo l'ultimo fallimento. Il valore atteso di questa misura prende il nome di *Mean Time to Repair* (MTTR).
- *Coverage*: Probabilità che il sistema sia in grado di tollerare un guasto.

La *safety* è un'estensione del concetto di *reliability*.

Si definisce uno stato sicuro in cui il sistema:

- Fornisce il servizio corretto, oppure
- Non fornisce il servizio corretto, ma il fallimento non ha conseguenze catastrofiche sull'ambiente o sulle persone.

Qualunque fallimento che induca il sistema a fornire un servizio scorretto con conseguenze catastrofiche, viene modellato come una transizione verso uno stato non sicuro. Quando esiste questa possibilità, il sistema viene definito *safety-critical*. [2]

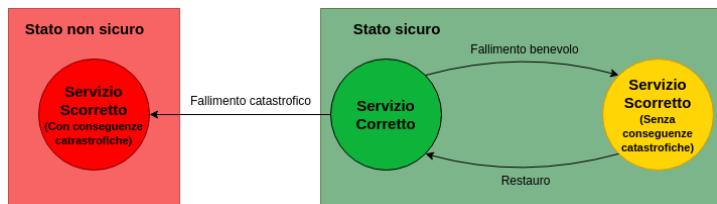


Figura 2: La *safety* estende il concetto di *reliability*

Esistono numerosi contesti in cui i sistemi impiegati vengono definiti *safety-critical*, uno di questi è il trasporto ferroviario.

I *threats* che minano la *dependability* di un sistema sono i guasti, gli errori e i fallimenti.

Un guasto è un qualunque evento interno al sistema in grado di causare un errore. Quando l'errore raggiunge l'interfaccia di servizio, ovvero altera il servizio fornito dal sistema, si parla di fallimento. In letteratura, si fa riferimento a questo rapporto di causalità come *fault error failure chain*, catena guasto errore fallimento.



Figura 3: Catena guasto errore fallimento

La *dependability* di un sistema informatico è raggiunta attraverso l'uso di quattro tecniche:

- *Fault Prevention*: previene l'insorgenza di guasti durante il ciclo di vita del sistema;
- *Fault Tolerance*: rende il sistema in grado di fornire un servizio corretto anche in presenza di guasti;
- *Fault Removal*: riduce il numero di guasti nel sistema;
- *Fault Forecasting*: stima il numero di guasti presenti nel sistema, attualmente o in futuro.

La *dependability* è la proprietà che viene misurata durante il processo di *validazione*.

La validazione è un processo atto a determinare se il sistema è conforme alle sue specifiche funzionali.

Il processo di validazione avviene durante tutte le fasi del ciclo di vita del sistema, anche prima che venga realizzato [11]. Per questo motivo, il sistema viene opportunamente modellato al fine di condurre propriamente l'analisi.

A discrezione della fase del ciclo di vita del sistema, si utilizzano differenti tecniche e modelli di validazione: [3]

- Specifica: la validazione è basata sull'utilizzo di tecniche combinatorie che mirano a determinare le condizioni di fallimento di un sistema in funzione del fallimento dei suoi sottocomponenti, considerati indipendenti;
- *Design*: in questa fase si usano modelli analitici *state-based* basati su processi casuali, come ad esempio le Catene di Markov. Si rilassano le assunzioni di indipendenza tipiche dei modelli combinatori;
- Implementazione: con il procedere della fase di implementazione, il sistema prende forma e può essere interessante osservarne

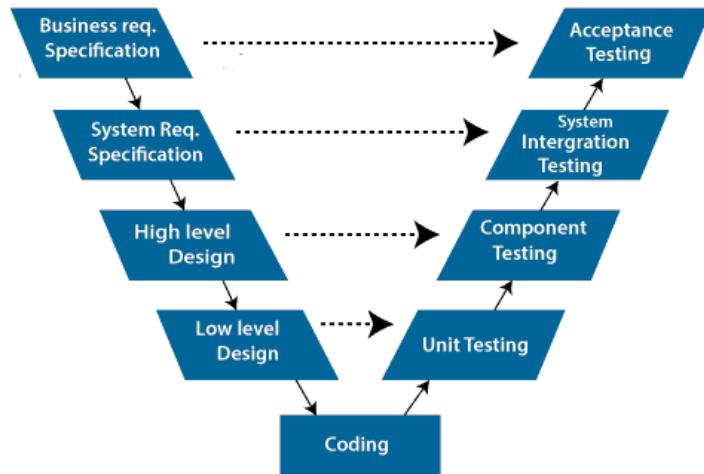


Figura 4: Il ciclo di vita dei sistemi: modello a V

direttamente il comportamento per effettuare misure sperimentali. L'attività di osservazione e misurazione prende il nome di *monitoring*;

- Fase operativa: il sistema è impiegato sul campo e possono essere utilizzate tutte le tecniche disponibili per l'analisi.

1.2 OSSERVAZIONE DEI SISTEMI

L'osservazione, o *monitoring*, è una tecnica per valutare la *dependability* di un sistema o un suo prototipo osservandone l'esecuzione nell'ambiente finale.

In questa sezione viene esposto il *monitoring* dei sistemi informatici come descritto nei lavori [4], [5] e [6].

L'obiettivo è quello di monitorare costantemente l'esecuzione del sistema, verificando che il comportamento osservato sia conforme alle aspettative. Un'attività di *monitoring* è seguita da un'attività di verifica. Questa può essere fatta durante l'esecuzione del sistema oppure *offline*, esaminando successivamente i risultati.

Il *monitoring* è stato riconosciuto come una valida opzione per valutare gli attributi di *dependability* dei sistemi informatici.

I problemi tecnici da affrontare e risolvere quando si crea un sistema di *monitoring* riguardano:

- Individuazione e classificazione degli eventi di interesse, al fine di raccogliere misurazioni utili per valutare correttamente la

dependability del sistema monitorato;

- Trasmissione delle informazioni al luogo dove queste saranno elaborate;
- Filtraggio e classificazione degli eventi rispetto alle misure di interesse.

Si definisce *target system* il sistema al quale vengono applicate le attività di *monitoring*. Il componente hardware o software interno al sistema verso il quale si riferisce il *monitoring*, viene detto *target component* o *target application*.

Esistono due differenti configurazioni di un processo di monitoring: *black-box* o *white-box*. Nella configurazione a *black-box* il sistema viene sottoposto a un certo carico di input al fine di osservare l'output prodotto. L'input fornito al sistema prende il nome di *workload*, ed è il carico di informazioni che il sistema dovrà processare durante il *monitoring*.

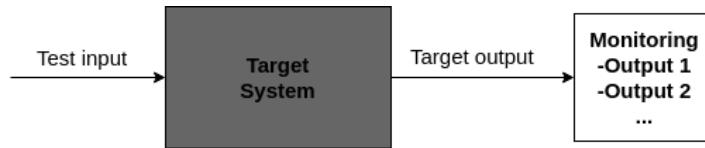


Figura 5: Monitoring black-box

Il monitoring *white-box* prevede l'utilizzo aggiuntivo di strumenti necessari a osservare anche gli output intermedi che il sistema produce. Questi strumenti sono chiamati sonde, o *probes*.

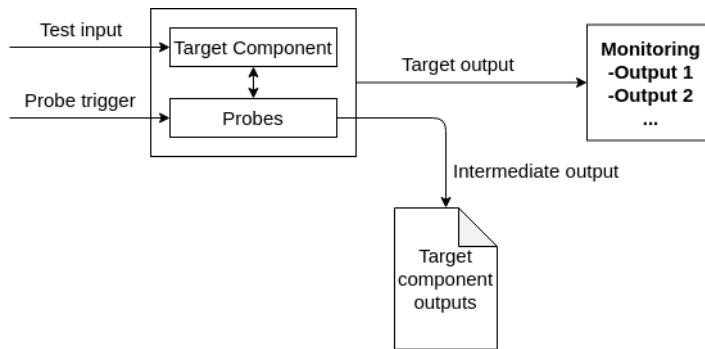


Figura 6: Monitoring white-box

I *probes* sono collocati solitamente all'interno del sistema e il loro scopo è quello di fornire informazioni più dettagliate sul comportamento del

target component.

Se si vogliono monitorare segnali hardware interni al sistema, i *probes* utilizzati saranno effettivamente hardware, mentre si utilizzano *probe* software se si vuole ottenere informazioni sull'esecuzione interna di un programma. Questa tecnica è chiamata instrumentazione del codice, e il codice aggiunto prende il nome di *software probe*.

Quando si progetta il *monitoring* di un sistema, si devono rispettare due regole principali:

- I *probes* devono essere in grado di raccogliere il giusto numero di informazioni circa il comportamento del sistema;
- Il comportamento del *target system* non deve essere alterato in maniera significativa a causa dell'inserimento dei *probes*. Quando questo avviene, si dice che il sistema di monitoraggio è *poco intrusivo*.

1.3 FAULT INJECTION

La *fault injection* è un approccio all'analisi della *dependability* che estende le tecniche di *monitoring* esposte in 1.2.

È definita come la volontaria introduzione di guasti all'interno di un sistema, al fine di osservarne il comportamento in presenza di guasti. [7] [8] [9]

Gli strumenti utilizzati per effettuare un'attività di *fault injection* sono mostrati in figura 7.

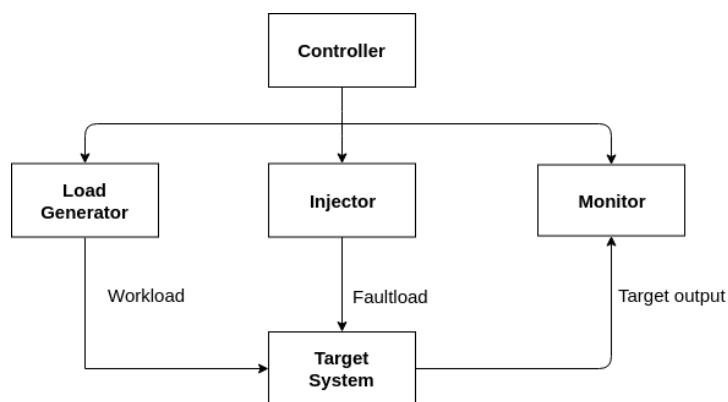


Figura 7: Elementi di un processo *fault injection*

Il *target system* è, come nel *monitoring*, il sistema che si vuole osservare. Il sistema viene alimentato con un *workload* generato dal *load generator*.

L'input fornito al sistema è costituito da dati che vengono normalmente processati dal sistema a *runtime*.

L'*injector* introduce guasti all'interno del sistema durante l'esperimento, i quali alterano la struttura o lo stato del sistema. Il carico di guasti iniettati prende il nome di *faultload*.

Il *monitor* colleziona i dati prodotti dal sistema per compiere misure di *dependability*, in maniera non diversa da quanto avviene nel classico *monitoring*.

Il *controller* coordina le operazioni degli altri componenti e itera gli esperimenti.

Da un punto di *dependability means*, la *fault injection* è una tecnica di *fault removal* e di *fault forecasting* in quanto viene testata l'efficacia dei meccanismi di *fault tolerance* del sistema e stimata la *dependability* osservando il comportamento dello stesso in presenza di guasti.

Le caratteristiche dei guasti iniettati, in termini di *tipo* e *frequenza*, vengono chiamate *fault model*.

Il *fault model* di un sistema è ottenuto considerando:

- I requisiti di sistema;
- L'ambiente in cui esso opera;
- Le tecnologie con cui il sistema è realizzato.

Per essere affidabile, un'attività di *fault injection* deve rispettare alcune proprietà fondamentali:

- Rappresentatività: sia il *workload* che il *faultload* devono corrispondere ai profili di utilizzo del sistema e alle modalità di fallimento durante la fase operazionale;
- Non intrusività: come nel caso del *monitoring*, l'intrusività data dall'iniezione di guasti e dall'osservazione del sistema deve essere minimizzata;
- Fattibilità: le tecniche di iniezione dei guasti devono richiedere un minimo sforzo implementativo, al fine di limitare i costi;
- Ripetibilità: Una campagna di *fault injection* deve produrre risultati comparabili se l'esperimento è ripetuto diverse volte sotto le stesse condizioni, per aumentare l'affidabilità dei risultati ottenuti.

Le attività di *monitoring* e *fault injection* fanno spesso affidamento sull'utilizzo di *tool* specifici. [10]

1.4 SCOPO DELLA TESI

Questa Tesi descrive l'attività di *monitoring* e *fault injection* effettuata sul prototipo di un sistema informatico impiegato nell'ambito del posizionamento ferrotramviario. Per quanto esposto in 1.1, il dominio ferrotramviario è un contesto *safety-critical*, come tale, lo sviluppo di sistemi informatici da impiegare nell'ambito ferrotramviario è regolamentato da specifici standard e requisiti di *dependability*.

Si introducono i sistemi ferrotramviari come derivazione dai classici sistemi ferroviari, si descrive il problema del posizionamento e le tecniche attualmente utilizzate per risolverlo, inquadrando il contesto operativo in cui si colloca il sistema analizzato.

2

STATO DELL'ARTE

2.1 SISTEMI FERROVIARI E FERROTRAMVIARI

È possibile schematizzare un sistema ferroviario, o ferrotramviario, come un insieme di vetture vincolate a muoversi lungo una traccia fissata. Questa schematizzazione è approssimativamente valida per qualsiasi sistema ferroviario o ferrotramviario, a prescindere dal numero di veicoli transitanti o dall'estensione geografica. Ciò che invece differenzia un sistema ferroviario da un sistema ferrotramviario sono:

- Le caratteristiche fisiche del veicolo transitante, come lunghezza e massa;
- Le caratteristiche geografiche dell'ambiente operativo;
- Gli scopi del trasporto.

In generale, nel trasporto ferroviario si utilizzano veicoli pesanti atti a trasportare persone o merci su lunghe percorrenze, pertanto è comune che l'ambiente operativo di un sistema ferroviario sia prevalentemente extra urbano.

Nel trasporto ferrotramviario, di contro, si utilizzano veicoli leggeri per offrire un'alternativa al cittadino all'utilizzo di mezzi privati durante i suoi spostamenti all'interno di un'area metropolitana. Quest'ultima caratteristica implica che l'ambiente operativo di un sistema ferrotramviario sia radicalmente diverso dall'ambiente operativo di un sistema ferroviario. Le vetture, anche dette *rotabili*, si muovono lungo tracce installate su strade urbane, e di conseguenza il traffico ferrotramviario condivide l'ambiente con il traffico cittadino, come mostrato in figura 8.



Figura 8: Schema di un tipico scenario ferrotramviario

2.1.1 *La safety nei sistemi ferroviari*

La gestione della *safety* nei sistemi ferroviari è regolamentata dallo standard EN 50126. [11]

Questo standard regolamenta la gestione della *safety* nel dominio ferroviario, con particolare enfasi verso le attività di valutazione RAMS (*Reliability, Availability, Maintainability, Safety*).

Esso prevede che vengano redatti i seguenti documenti.

SAFETY PLAN

Un insieme documentato di attività programmate, risorse ed eventi necessari a implementare la struttura organizzativa, le responsabilità, le procedure, le attività, le funzioni e le risorse che insieme assicurano la *safety* del sistema.

In sintesi, il *safety plan* individua *chi fa cosa, e quando*, al fine di realizzare un sistema *safe*.

SAFETY REQUIREMENTS

Definisce i requisiti di *safety* che il sistema deve soddisfare prima di poter essere impiegato sul campo.

SAFETY CASE

La dimostrazione documentata che un prodotto sia conforme ai suoi requisiti di *safety*. Il *Safety Case* deve essere redatto come una dimostrazione

logica della *safety* del sistema.

Per quanto riguarda le procedure e i requisiti tecnici per lo sviluppo di applicazioni software utilizzate nel controllo ferroviario, lo standard di riferimento è EN 50128. [12]

Lo standard EN 50128 impone che a ciascun software integrato all'interno di un sistema ferroviario debba essere assegnato un *Safety Integrity Level* (SIL). [13]

Il SIL viene assegnato sulla base del tempo medio al fallimento catastrofico, ossia il MTTF relativo a fallimenti catastrofici.

Livello SIL	Tempo medio al fallimento catastrofico
SIL-1	$[10^5 - 10^6]$ ore
SIL-2	$[10^6 - 10^7]$ ore
SIL-3	$[10^7 - 10^8]$ ore
SIL-4	$\geq 10^8$ ore

Tabella 1: Livelli SIL

Mentre lo standard EN 50126 è generico e applicabile a qualunque sistema ferroviario o parti di esso, poiché relativo alla *gestione* della *safety* piuttosto che al suo effettivo raggiungimento, lo standard EN 50128 è inequivocabilmente tecnico e specifico per i software critici. Esso identifica le fasi principali nel ciclo di vita del software e per ciascuna di queste prevede un insieme di tecniche da utilizzare per soddisfare il SIL assegnato al software.

2.2 IL PROBLEMA DEL POSIZIONAMENTO

Per posizionamento ferroviario si intende la valutazione della posizione di un treno all'interno di una traccia ferroviaria. Tale posizione viene espressa come progressiva chilometrica rispetto a una posizione nota, come ad esempio l'origine della linea. [14]

2.2.1 Tradizionali tecniche di posizionamento

I tradizionali sistemi di posizionamento si basano principalmente sull'utilizzo di strumenti installati a terra, chiamati *beacon*, o *balise* in gergo

ferroviario, i quali hanno lo scopo di rilevare il passaggio di un treno. [15]

Esiste uno standard a livello europeo al quale i tradizionali sistemi di posizionamento si devono uniformare, l' *European Train Control System* (ETCS).

Nel corso della storia, ogni paese europeo ha sviluppato autonomamente le proprie infrastrutture ferroviarie e relative regole operative. Tuttavia, ad oggi i treni possono attraversare le frontiere, pertanto è necessario sviluppare un sistema ferroviario standard che rispetti una comune normativa operazionale europea. Tale sistema prende il nome di *European Rail Traffic Management System* (ERTMS) [16], ed ETCS è il sottosistema di ERTMS dedicato al posizionamento delle vette.

Come standard, ETCS definisce specifici livelli di *compliance* che possiede un sistema di posizionamento rispetto ad ETCS, ed essi vanno dal livello ETCS - 0 al livello ETCS - 3.

L'obiettivo è quello di sviluppare progressivamente un sistema di posizionamento completamente autonomo (ETCS - 3), partendo da un sistema interamente *non-compliant* con ETCS (ETCS - 0).

Allo stato attuale, quasi tutti i sistemi di posizionamento sono ETCS - 2. Nei livelli ETCS - 1 e ETCS - 2, le tracce vengono suddivise in blocchi, e all'entrata di ciascun blocco viene posizionato un *beacon* in grado di rilevare la presenza di un treno.

L'autorizzazione all'ingresso in un blocco viene rilasciata se nessun altro treno sta occupando il blocco al quale si vuole accedere, mentre un sistema di *odometria* installato a bordo, posiziona il treno rispetto all'ultimo *beacon* incontrato.

Nel livello ETCS - 3, non sono richiesti segnali provenienti dalla linea: un treno deve essere in grado di posizionarsi autonomamente. [17]

In sintesi, i livelli ETCS possono essere descritti come segue:

- ETCS - 0: Sistema non conforme a ETCS;
- ETCS - 1: Utilizzo di apparati di posizionamento installati a terra, autorizzazione a procedere segnalata al macchinista attraverso indicazioni semaforiche;
- ETCS - 2: Come ETCS - 1, ma l'autorizzazione a procedere è gestita da un sistema automatico di scambio, denominato sistema di *interlocking*; [18]
- ETCS - 3: Posizionamento autonomo, nessun utilizzo di apparati a terra.

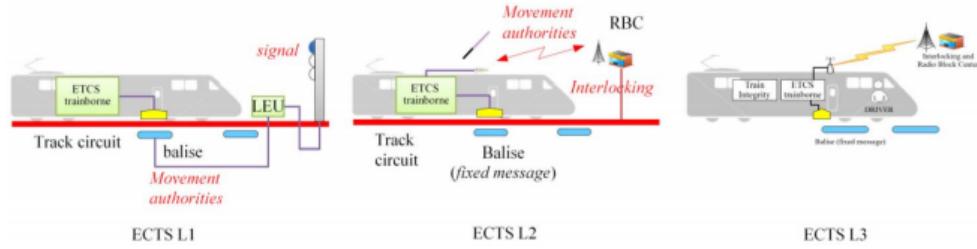


Figura 9: Livelli ETCS

Il livello ETCS-2 prevede che l'autorizzazione a procedere venga gestita dal sistema di *interlocking* e non dal solo operatore umano notificato mediante indicazioni semaforiche.

La funzionalità offerta del sistema di *interlocking* viene pertanto considerata *safety-critical*, in quanto un suo fallimento può portare a conseguenze anche catastrofiche. [19]

ETCS adotta un approccio incrementale alla realizzazione di sistemi di posizionamento autonomi. Un sistema di posizionamento ETCS-3 deve continuare a interagire con il sistema di *interlocking*, quindi deve essere considerato a sua volta un sistema *safety-critical*.

ERTMS/ETCS è pensato per sistemi ferroviari, mentre nel dominio ferrotramviario vige la regola della *marcia a vista*. Il rispetto di ETCS non è obbligatorio in detto contesto, tuttavia le tecniche di posizionamento ivi utilizzate rispettano spesso le linee guida imposte da ETCS.

2.2.2 Verso ETCS-3

Gli attuali sistemi di posizionamento richiedono un minimo intervento di computer installati a bordo e una grande quantità di apparati installati a terra. Gli apparati di terra sono costosi e hanno un impatto ambientale non trascurabile, pertanto è necessario iniziare a pianificare una migrazione verso sistemi ETCS-3. [20]

Il sistema analizzato in questa Tesi è un sottosistema di posizionamento conforme alla filosofia ETCS-3. Nell'ottica di migrazione verso sistemi ETCS-3, è stato progettato un sistema di posizionamento ferrotramviario autonomo, basato principalmente sull'utilizzo di un *sensore inerziale*, un odometro e un GPS installati a bordo treno, le cui misurazioni vengono processate da un software al fine di stimare la posizione del treno. [21] Le misure fornite al software vengono integrate attraverso l'utilizzo di

un algoritmo noto come *Sensor Fusion Algorithm* (SFA). [22]

Attualmente il software ha superato le fasi di analisi e definizione dei requisiti, *system design*, e si trova nella fase di implementazione. Si è quindi reso disponibile per l'analisi di *dependability* un prototipo del software. Per quanto esposto in 1.1, questo può essere osservato durante l'esecuzione nel suo ambiente operativo.

L'analisi condotta sul sistema è del tipo *fault injection*. Poichè esso opererà in un contesto *safety critical*, il focus è quello di valutare l'efficacia dei meccanismi di *fault tolerance* implementati nel sistema.

Il *target system* è l'intero sottosistema di posizionamento installato a bordo, quindi include gli strumenti di misura e l'*On Board Control Unit* (OBCU). OBCU è un computer di bordo che dovrà elaborare le informazioni sulla posizione del treno al fine di interagire, quando necessario, con il sistema di *interlocking* della traccia.

Il *target component* in esame è il modulo software che processa le misure campionate. Durante il processo di analisi, si farà affidamento a un *tool* realizzato appositamente per valutare le performance del software che implementa SFA. Il *tool* utilizzato include:

- Un *load generator* che alimenta SFA con misure che verosimilmente potrebbero essere campionate dal sistema nel suo ambiente reale;
- Un *injector* in grado di iniettare i guasti che potrebbero verificarsi quando il sistema sarà impiegato sul campo;
- Un *monitor* che colleziona le uscite e i risultati intermedi del modulo SFA.

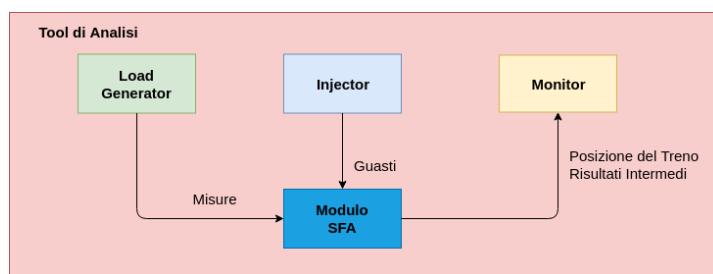


Figura 10: Schema dell'ambiente di *fault injection*

Il *tool* si comporta come *controller* in quanto è in grado di controllare il *load generator*, l'*injector* e il *monitor*. Esso itera gli esperimenti un numero arbitrario di volte, stabilito dall'osservatore, e fornisce un report riassuntivo della campagna sperimentale. Il modulo SFA è incluso nel *tool* come libreria.

2.3 CYBER PHYSICAL SYSTEMS OF SYSTEMS

Il sottosistema di posizionamento studiato rientra nella categoria dei *Cyber Physical Systems of Systems* (CPSoS). [23]

Un *Cyber Physical System* (CPS) è un sistema composto da due parti:

- *Cyber*: Elementi di computazione, comunicazione e controllo che operano in un tempo discreto;
- *Physical*: Sottosistema naturale o costruito dall'uomo che è governato dalle leggi della fisica e opera in un tempo continuo.

Un CPS è un sistema che consiste in un computer (il sistema *cyber*) e un oggetto controllato (il sistema *physical*), che include la possibilità di interazione con gli esseri umani.

In talune applicazioni i sistemi sono integrati in un più ampio contesto al fine di fornire un *servizio* che un sistema *standalone* non sarebbe in grado di fornire. Si parla in questo caso di *Systems of Systems* (SoS).

I SoS sono caratterizzati da una gerarchia multi-livello: una struttura ricorsiva in cui un sistema viene considerato *il tutto* al più alto livello di interesse (*macro-level*), e composto da un insieme di sottosistemi al livello sottostante (*micro-level*). Ciascun sottosistema può essere considerato a sua volta come un sistema, e la ricorsione termina quando l'ulteriore suddivisione di un sottosistema perde di interesse.

Si definisce *componente* un sottosistema al più basso livello di interesse.

Un SoS composto da CPS prende il nome di CPSoS.

Gli elementi architetturali che compongono un CPSoS sono:

- *Itom*: costituiscono l'unità minima di interazione. Sono elementi informativi composti da dati e metadati;
- *Constituent Systems* (CS): Entità di tipo CPS che processano e scambiano gli *itom*;
- Ambiente: tutte le entità con cui può interagire un CS.

I CS interagiscono fra di loro, o con l'ambiente, scambiandosi *itom* attraverso le loro *interfacce*.

Si distinguono le seguenti interfacce:

- *Local I/O* (L-interface): interfacce che abilitano l'interazione tra un CS e il suo ambiente. Non sono rilevanti ai fini del servizio che si intende fornire;

- *Configuration* (C-interface): interfacce utilizzate per scopi di configurazione o *update* hardware e software;
- *Diagnostic* (D-Interface): Interfacce di diagnostica utili per il *monitoring* di un CS;
- *Time-Sync* (TSI): interfacce di sincronizzazione esterna che vengono utilizzate per rendere un CS *time-aware* e stabilire una *global timebase*;
- *Relied Upon Interfaces* (RUI): Un CPSoS fa affidamento alle RUI per fornire il servizio atteso.

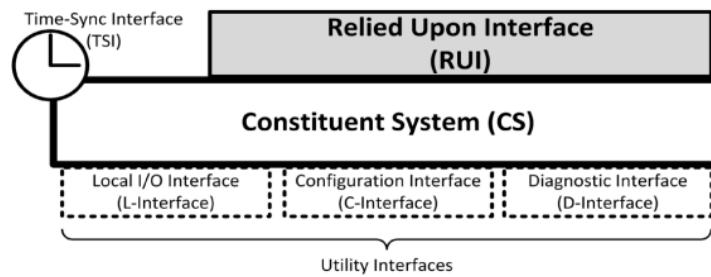


Figura 11: Interfacce di un CPS

La specifica delle RUI è di particolare importanza poiché qualunque struttura del sistema, responsabile del comportamento osservato, può essere ridotta alla specifica delle interfacce del sistema. [32]

In base al canale di comunicazione e agli *item* scambiati attraverso le RUI, si distinguono:

- *Relied Upon Message Interfaces* (RUMI): L'interazione consiste di uno scambio di messaggi a livello *cyber*, come ad esempio un pacchetto TCP;
 - *Relied Upon Physical Interfaces* (RUPI): Le componenti *physical* di un CS interagiscono attraverso un canale *stigmergico*.
- Un CS altera una grandezza fisica nell'ambiente, in maniera tale per cui un altro CS possa trarre contenuto informativo attraverso il rilevamento della modifica.

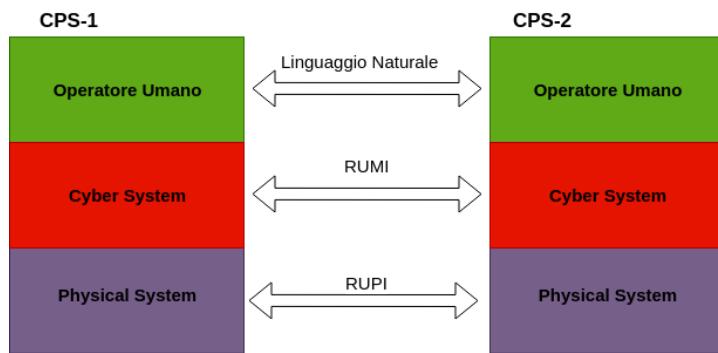


Figura 12: RUMI e RUPI

Un CPS crea canali stigmergici attraverso l'uso di *sensori* e *attuatori*. I sensori sono in grado di rilevare parte delle grandezze fisiche che caratterizzano l'ambiente operativo di un CS, mentre gli attuatori sono in grado di alterare tali grandezze.

3

IL SISTEMA ANALIZZATO

3.1 DESCRIZIONE GENERALE

Il sistema analizzato rientra nella categoria dei CPSoS. Lo scopo del sistema è quello di implementare un meccanismo di posizionamento basato su SFA.

Tale algoritmo viene eseguito da una libreria software schematizzabile, ai fini di questa Tesi, come una *black-box* che rappresenta il nucleo centrale del sottosistema di posizionamento.

Ricevuti in ingresso un certo insieme di misure, essa fornisce in uscita una *stima statistica* della posizione del treno, più accurata della misura che si otterrebbe utilizzando i dati provenienti dai singoli sensori. [24]
La posizione viene fornita sia in termini di progressiva chilometrica che di coordinate geografiche.

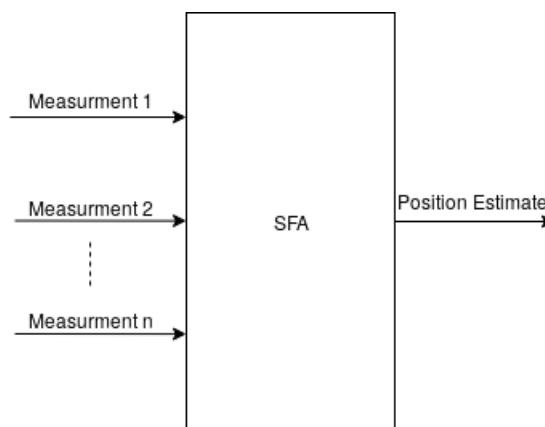


Figura 13: Schema SFA

SFA viene eseguito su di un hardware installato a bordo treno, e la sua esecuzione è volta a monitorare costantemente il moto del treno.

Il ciclo di esecuzione di SFA è essenzialmente una continua iterazione di due distinte fasi logiche:

- Acquisizione delle misure;
- Predizione della posizione del treno.

Le grandezze fisiche che dovranno essere misurate e fornite a SFA sono:

- Vettore accelerazione;
- Vettore velocità angolare;
- Coordinate geografiche;
- Velocità lineare (scalare).

In quest'applicazione, SFA utilizza tali informazioni in combinazione con un'apposita digitalizzazione della traccia tramviaria su cui si trova il treno. [25][26][27]

Queste informazioni si suppongono note a priori ed accedibili tramite un *database* caricato in memoria centrale. [28]

Nel database occorre predisporre una lista di punti geometrici appartenenti alla traccia, caratterizzati dalle loro coordinate e dalla progressiva chilometrica associata. Il sistema modella la traccia come una *spline* interpolante i punti selezionati.

3.2 CONSTITUENT SYSTEMS

Il sottosistema di posizionamento si compone dei moduli, o CS, descritti nel seguito di questa sezione.

3.2.1 Sensor Set

Il *Sensor Set* è un insieme di sensori atti a campionare le misure richieste da SFA. Esso si compone a sua volta dei seguenti moduli:

- *Inertial Measurement Unit* (IMU):
Sensore inerziale incaricato di campionare e trasmettere a SFA i vettori accelerazione (\mathbf{a}) e velocità angolare (\mathbf{v}_{ang}). Le misure di

IMU sono prese rispetto alla Terra e sono espresse in unità stabilite dallo standard internazionale (SI):

$$\mathbf{a} \left[\frac{\text{m}}{\text{s}^2} \right] \quad \mathbf{v}_{\text{ang}} \left[\frac{\text{rad}}{\text{s}} \right]$$

IMU è il sensore principale su cui si basa SFA nel predire la posizione del treno. Date le caratteristiche intrinseche del particolare SFA utilizzato, ossia un *Filtro di Kalman*, il sistema funziona anche senza i rimanenti sensori. Si osserverebbe tuttavia un calo delle performance in termini di errore commesso sulla stima della posizione del treno. [29] [30]

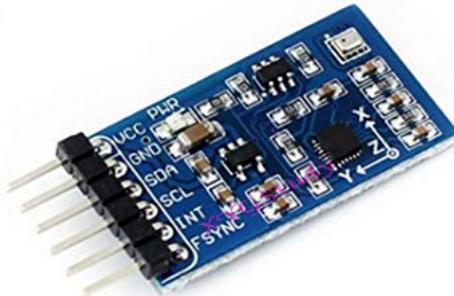


Figura 14: *Inertial Measurement Unit*

- Odometro:
Unità incaricata di fornire a SFA i valori di velocità lineare del treno, espressi in $\frac{\text{m}}{\text{s}}$.
- GPS:
Unità che fornisce a SFA le misure di posizione del treno.
Le misure di GPS sono riportate in formato standard come tripla di coordinate (*latitudine*, *longitudine*, *altitudine*), rispettivamente espresse in gradi N-S, in gradi E-O e in metri sul livello del mare.

3.2.2 Piattaforma di elaborazione dati

La piattaforma di elaborazione dati è l'hardware sul quale viene eseguito SFA. Consiste di una scheda Nvidia TX-Jetson collegata al *Sensor Set*. Da quest'ultimo essa riceve le misure da processare tramite SFA. Nvidia



Figura 15: Ricevitore GPS ublox EVK-M8T



Figura 16: Nvidia TX-Jetson

TX-Jetson è un’architettura specifica per sistemi *embedded*. Essa è ottimizzata per i calcoli computazionalmente onerosi tipici delle applicazioni di intelligenza artificiale. [31]

3.2.3 OBCU

L’OBCU è il computer di bordo del treno. Esso non svolge alcun ruolo attivo nel sistema di posizionamento, tuttavia la progressiva chilometrica, stimata da SFA, dovrà essere trasmessa a OBCU al fine di poter utilizzare questa informazione all’interno del sistema di *interlocking* della traccia.

GPU	256-core NVIDIA Pascal
CPU	Dual-Core NVIDIA Denver 2 64-Bit CPU Quad-Core ARM Cortex-A57 MPCore
Memoria	8GB 128-bit LPDDR4 Memory
Storage	32GB eMMC 5.1
Alimentazione	7.5W / 15W

Tabella 2: Specifiche Tecniche NVidia TX-Jetson

3.3 SPECIFICA DELLE INTERFACCE

3.3.1 RUII

In questa sezione si evidenziano le principali interfacce del sistema, alle quali si osservano le interazioni fondamentali che avvengono al suo interno.

Il sistema di posizionamento interagisce con il treno attraverso le RUPI del *Sensor Set*, ossia gli strumenti di misura che esso integra. Questi sensori campionano, a diverse frequenze, i dati sul moto del treno che verranno elaborati da SFA. Una descrizione sintetica delle RUPI del sistema è mostrata in tabella 3.

RUPI	Grandezza Campionata	Parti interagenti
Accelerometro	Accelerazione	Vettura - IMU
Giroscopio	Velocità angolare	Vettura - IMU
Radar	Velocità lineare	Vettura - Odometro
Ricevitore GPS	Coordinate geografiche	Vettura - GPS

Tabella 3: Specifica delle RUPI del sistema

Per quanto concerne le RUMI, se ne osservano di due tipi:

- Tre bus dati, che collegano il *Sensor Set* alla scheda Nvidia TX-Jetson. Su ciascuno di essi, *Sensor Set* invia rispettivamente messaggi contenenti i dati campionati da IMU, Odometro e GPS.
- Interfaccia LTE. Essa permette di realizzare una *rete wireless ad hoc* fra la scheda e OBCU.
All'interno di tale rete vengono instradati datagrammi UDP conte-

nenti le informazioni sulla progressiva chilometrica stimata da SFA, ed eventualmente messaggi di *acknowledgment* di OBCU verso la scheda.



Figura 17: Modem TP-LINK M7350 LTE-4G

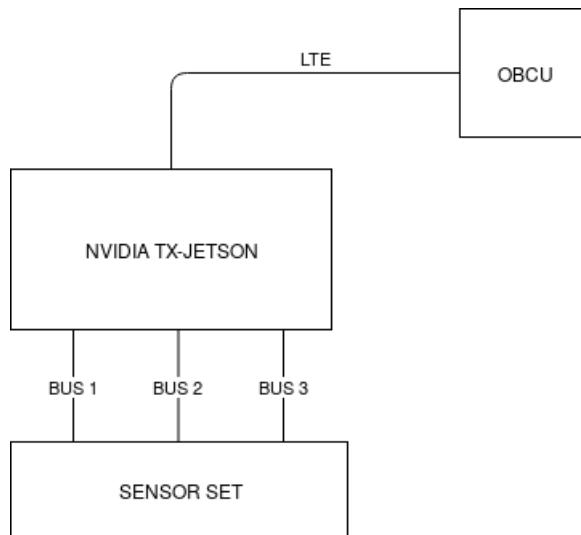


Figura 18: Diagramma a blocchi del sottosistema di posizionamento

3.4 INTERAZIONI

In questa sezione si descrivono le interazioni osservabili alle interfacce sopra descritte. Queste possono essere in prima istanza categorizzate a discrezione della fase di SFA in cui esse avvengono.

Si distinguono pertanto le interazioni riguardanti l'acquisizione dei dati

RUMI	Informazione trasmessa	Parti interagenti
Bus Dati 1	Accelerazione, Velocità angolare	Sensor Set - NVidia TX-Jetson
Bus Dati 2	Velocità lineare	Sensor Set - NVidia TX-Jetson
Bus Dati 3	Coordinate geografiche	Sensor Set - NVidia TX-Jetson
LTE	Posizione del treno	NVidia TX-Jetson - OBCU

Tabella 4: Specifica delle RUMI del sistema

in ingresso a SFA, e le interazioni riguardanti l’acquisizione da parte di OBCU della posizione del treno.

3.4.1 Acquisizione dei dati

L’acquisizione dei dati si divide in due differenti interazioni: la prima, con la vettura, avviene alle RUPI del *Sensor Set*, mentre la seconda avviene alle RUMI bus dati che collegano il *Sensor Set* alla piattaforma di elaborazione dati.

I moduli che compongono il *Sensor Set* campionano ad una data frequenza le grandezze fisiche che descrivono il moto del treno. Ciascun campionamento fisico è seguito dall’invio dei valori letti alla piattaforma di elaborazione dati. I moduli del *Sensor Set* sono tra di loro indipendenti. In figura 19 viene riportato un *sequence diagram* rappresentante una possibile sequenza di campionamento e invio dei dati. Questa tipologia di interazione è detta *time-triggered*, in quanto è determinata unicamente dallo scorrere del tempo. [34]

3.4.2 Trasmissione della posizione

Ogniqualvolta viene completato un aggiornamento di SFA, si osserva un’interazione all’interfaccia LTE. Tale interazione consiste nell’invio di un messaggio contenente la posizione del treno, dalla piattaforma di elaborazione dati verso OBCU, e nella trasmissione di un messaggio di *acknowledgment* nel senso opposto.

La tipologia di scambio dei messaggi esposta è detta *event-triggered* [35] in quanto le tempistiche di interazione non sono note a priori, ma dipendono dal tempo computazionale impiegato da SFA nell’aggiornare la propria stima della posizione.

LTE è a tutti gli effetti una regolare interfaccia di rete. Il messaggio

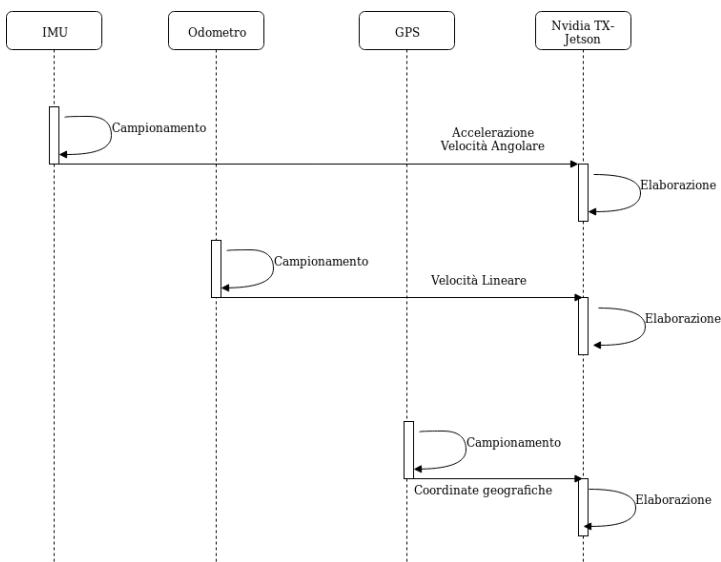


Figura 19: Sequenza di acquisizione dati

trasmesso è contenuto nel *payload* di un datagramma UDP.

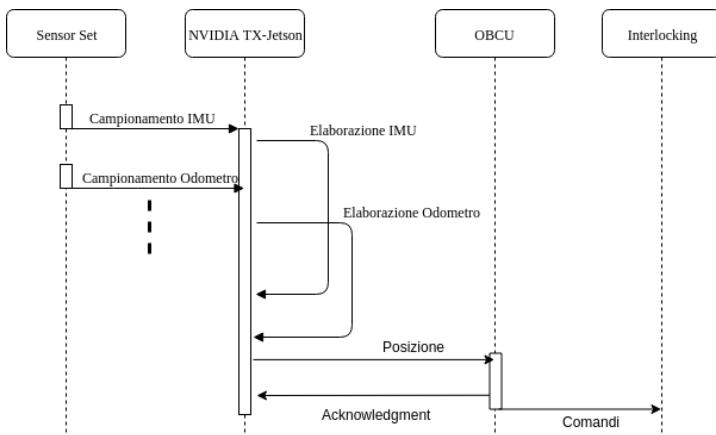


Figura 20: Sequenza di trasmissione della posizione

3.4.3 Interazioni con eventuali operatori umani

La filosofia del sistema include la minimizzazione di interventi da parte di operatori umani.

A questi viene lasciato il compito di predisporre le informazioni geografiche della traccia nel database, e di segnalare l'avvio al sistema.

Il processo che esegue SFA è in effetti un *demone* che si avvia contestual-

mente all'accensione della piattaforma.

Viene infine predisposta un interfaccia SSH per accedere da remoto alla piattaforma a scopi di *deployment* o di *monitoring online* attraverso la lettura dei file di *log*.

3.5 ARCHITETTURA SOFTWARE

In questa sezione viene completata la panoramica sul sistema descrivendo i moduli software in esecuzione sulla piattaforma NVidia TX-Jetson. Su tale piattaforma è installato il sistema operativo Ubuntu 16.04 LTS, basato su Kernel Linux.

3.5.1 *Sensor Fusion Library*

Il software che esegue SFA viene fornito come libreria, denominata *SensorFusionLib*, la quale mette a disposizione dei *client* le API descritte di seguito.

API

Le API di *SensorFusionLib* definiscono le interfacce software verso il modulo SFA che possono essere utilizzate dai *client*.

Le principali funzioni disponibili sono le seguenti:

- **FusionInit(args...)**
Funzione che inizializza SFA. Tale funzione deve essere chiamata quando è necessario avviare l'algoritmo. Essa riceve come parametri i valori che caratterizzano le condizioni iniziali del moto, come l'errore iniziale rispetto alla progressiva chilometrica e alla velocità;
- **ProcessInertialMeasurementData(args...)**
Funzione che permette a SFA di ricevere ed elaborare un campionamento di IMU;
- **ProcessOdometryMeasurementData(args...)**
Funzione che permette a SFA di ricevere ed elaborare un campionamento di Odometro;
- **ProcessGPSMeasurementData(args...)**
Funzione che permette a SFA di ricevere ed elaborare un campionamento di GPS;

- `ProcessStrobe(args...)`

Funzione che deve essere invocata ogni secondo, per permettere a SFA di sincronizzarsi rispetto a una *global timebase* esterna; [33]

- `IsUpdated()`

Funzione che restituisce vero se SFA ha completato un'iterazione ed è pronto a fornire l'output prodotto;

- `GetFusionOutput()`

Se `IsUpdated()` restituisce vero, è possibile invocare questa funzione per ricevere da SFA l'ultimo output calcolato.

3.5.2 Listener

SensorFusionLib è incapsulato all'interno di un eseguibile, *listener*.

Questo software possiede un'istanza di *SensorFusionLib* con la quale interagisce attraverso le API descritte in 3.1.1. Esso dispone di due *socket* UDP, una utilizzata per ricevere i dati *raw* provenienti dai sensori, l'altra per la comunicazione remota con OBCU.

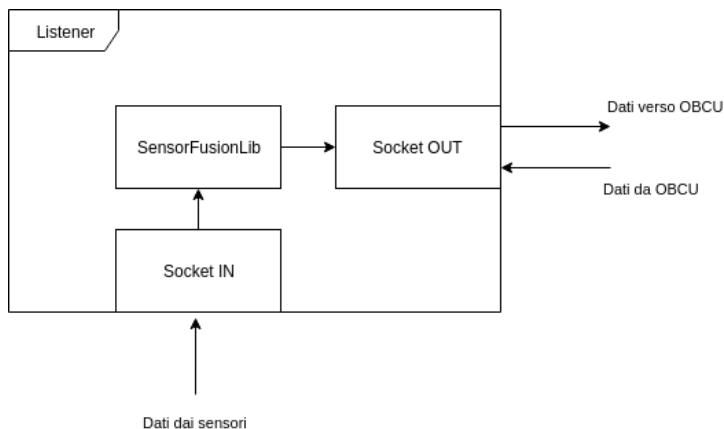


Figura 21: Diagramma a blocchi di *listener*

3.5.3 Interface Modules

La comunicazione con i sensori è gestita interamente da un set di processi denominati *interface-modules*. Ciascun sensore è collegato ad un'interfaccia seriale della piattaforma attraverso un bus dati. Per ogni interfaccia connessa, un processo resta in ascolto su di essa. Quando un sensore invia un campionamento su una specifica interfaccia, il processo in ascolto

su quest'ultima si fa carico di inoltrare a *listener* i valori ricevuti. Ciascun processo di *interface-modules* dispone di una *socket UDP* che abilita la comunicazione con *listener*.

Modulo	Sensore	Dati Inviai a <i>listener</i>
<i>IMU Process</i>	IMU	Accelerazione, Velocità angolare
<i>ODO Process</i>	Odometro	Velocità lineare
<i>GPS Process</i>	GPS	Coordinate Geografiche
<i>Strobe Process</i>	N/A	Sincronizzazione

Tabella 5: Moduli di *interface-modules*

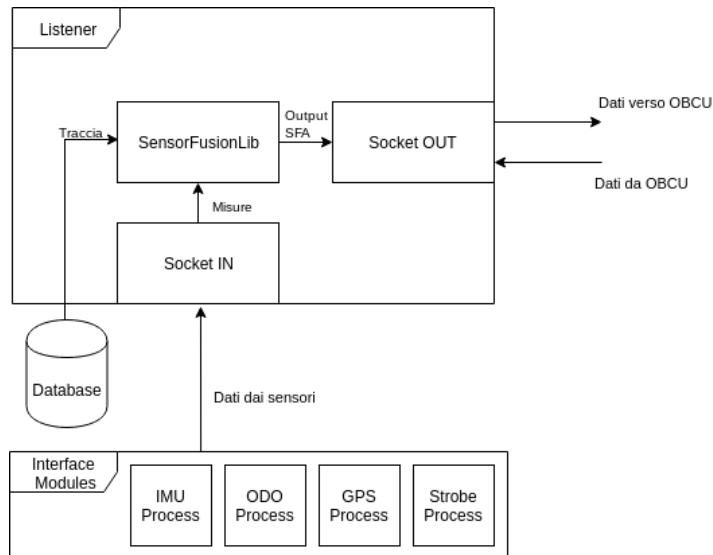


Figura 22: Diagramma a blocchi del sistema software

4

AMBIENTE DI ANALISI

Lo scopo dell’analisi è quello di effettuare misure di *dependability* sul software che implementa SFA (*SensorFusionLib*), attraverso una campagna di *fault injection*.

Il sistema su cui verrà effettuata l’analisi è un’architettura x86 equipaggiata con il sistema operativo Windows 7.

In questo capitolo si descrive l’ambiente di analisi mappando gli strumenti teorici, noti dalla letteratura ed esposti in 1.3, su quelli effettivamente utilizzati.

Il *target component* è la libreria *SensorFusionLib*.

4.1 IL TOOL DI ANALISI

Il *tool* di analisi utilizzato è un software denominato *RailTrackTool* (RTT). RTT è un software appositamente progettato per valutare le performance di SFA, effettuare campagne di *monitoring* e *fault injection*. Esso funge da *controller* ed integra al suo interno il *load generator*, l’*injector* e il *monitor*.

4.1.1 Load Generator

Il *load generator* è lo strumento preposto a fornire il *workload* al sistema. In quest’applicazione, SFA viene alimentato dalle misure campionate da IMU, Odometro e GPS, installati a bordo di un treno che si muove lungo una traccia ferrotramviaria.

Un ulteriore input è rappresentato dalle informazioni geografiche della traccia.

È stata realizzata una libreria software (*SynthDataGen*) in grado di generare le misure che verosimilmente verrebbero campionate nel sistema reale, in base ai riferimenti geografici della traccia.

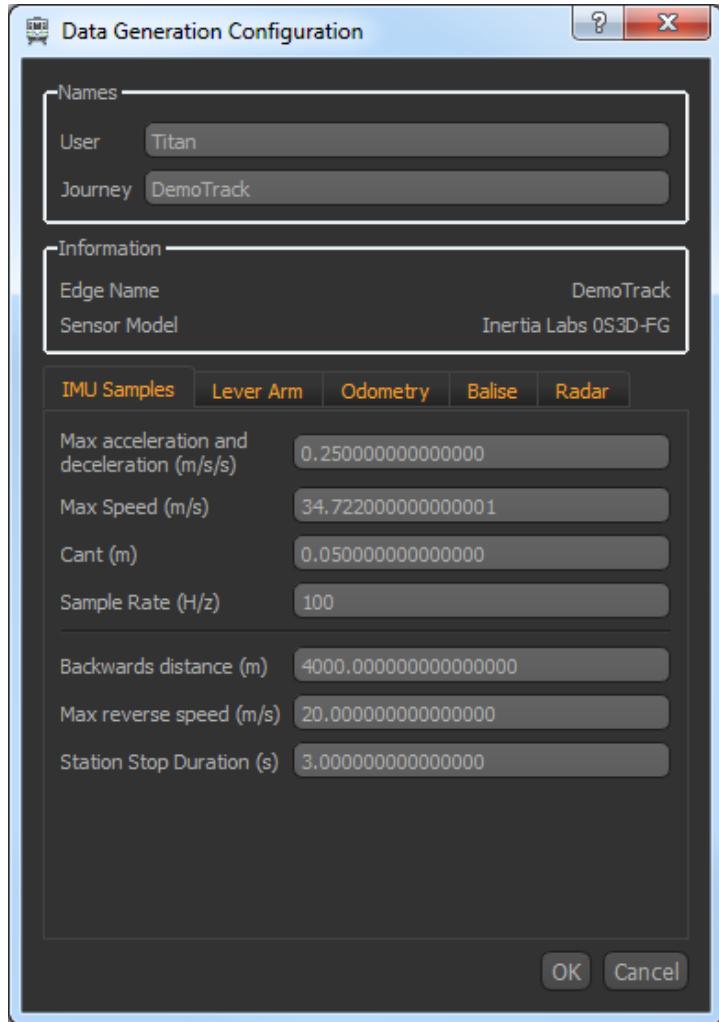


Figura 23: Interfaccia utente verso *SynthDataGen*

L'osservatore ha la possibilità di definire:

- Le caratteristiche tecniche dei sensori: rumore di misura [36] e frequenza di campionamento;
- La velocità massima che il treno può raggiungere;
- L'accelerazione massima.

Il rumore di misura viene fornito al *load generator* come:

- Matrice di covarianza, per campionamenti multivariati (IMU);
- Varianza, per campionamenti univariati (Odometro).

Quando viene generato un possibile campionamento, il valore prodotto viene perturbato di una quantità pari al valore generato da una *distribuzione gaussiana* a media nulla e varianza (o covarianza) specificata.

SynthDataGen non supporta la generazione di campionamenti GPS. Questo accade poiché i valori campionati da un GPS sono caratterizzati da un rumore di misura che non rispetta una distribuzione gaussiana, infatti la *reliability* delle informazioni GPS dipende da parametri non predicibili come il numero di satelliti attualmente in grado di fornire la posizione al ricevitore. [30]

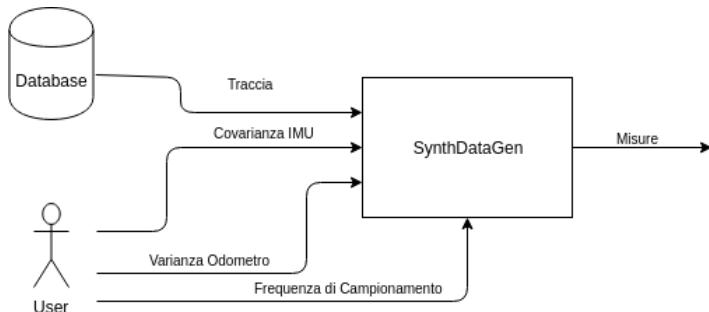


Figura 24: Diagramma a blocchi *SynthDataGen*

4.1.2 *Injector*

Il *faultload* è generato da uno specifico modulo (classe) di RTT. L'utente, attraverso l'interfaccia del software, è in grado di:

- Sopprimere, per un arbitrario periodo di tempo, il canale di comunicazione tra i sensori e il modulo SFA;
- Alterare in modo casuale il contenuto dei messaggi trasmessi dai sensori al modulo SFA;
- Modificare i riferimenti geografici della traccia forniti in ingresso al modulo SFA.

È stato individuato questo *fault model* considerando in primo luogo i requisiti del software. In particolare:

- Ogni campionamento ricevuto dal sistema **deve** essere indicizzato;
- Il sistema **deve** scartare un campionamento quando questo viene ricevuto fuori ordine o il suo valore non è considerato accettabile in relazione ai valori ricevuti fino a quel momento;

- Quando il sistema scarta un campionamento, **deve** sostituirlo con una predizione del contenuto corretto, attraverso regressione lineare.

In secondo luogo, è opportuno considerare le tecnologie utilizzate. In 3.5.3 è stato dichiarato che la comunicazione tra i sensori e il modulo SFA, nel sistema reale, avviene utilizzando UDP.

Questo protocollo di comunicazione non garantisce:

- La consegna dei messaggi;
- L'ordinamento dei messaggi;
- L'integrità dei messaggi.

4.1.3 Monitor

L'interfaccia utente di RTT fornisce una mappa su cui verrà marcata la posizione del treno durante gli esperimenti.

I risultati intermedi prodotti da SFA sono mostrati su un grafico e riportati in un file di *log*.

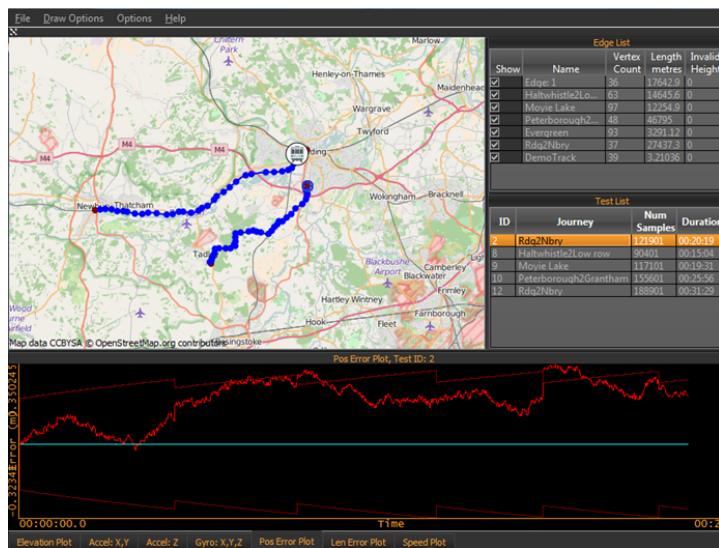


Figura 25: Interfaccia utente di RTT

4.1.4 Controller

RTT integra al suo interno tutti gli elementi di una campagna di *fault injection*, ed è pertanto in grado di controllarli.

Per aumentare la *reliability* dei risultati prodotti, RTT permette all'utente di definire il numero di ripetizioni da effettuare durante l'esecuzione di un esperimento. Al termine di ciascun esperimento, RTT processa il file

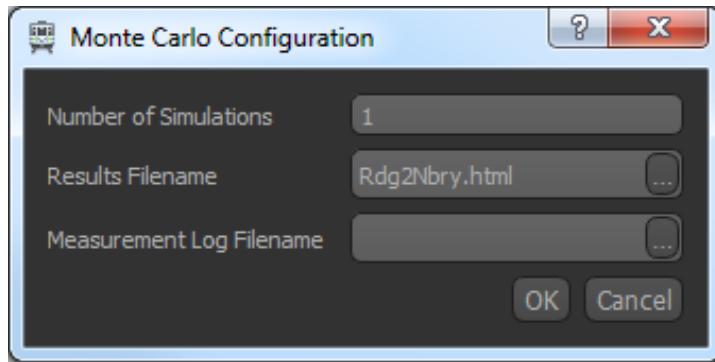


Figura 26: Controller RTT

di *log* prodotto dal modulo SFA e costruisce un report HTML contenente una sintesi dei risultati ottenuti (figura 28).

4.2 NOTE SULL'INTRUSIVITÀ

È noto dalla letteratura che un sistema di *monitoring*, così come di *fault injection*, deve essere il meno intrusivo possibile nei confronti del sistema

EPOCH	SENSOR	ImuCnt	Dropped	GT_cnt	GT_Edge	GT_Position	GT_Position_norm	GT_Speed	GT_Dir	GT_ecef_x	GT_ecef_y	GT_ecef_z	FU_cnt	FU_Edge	FU_Arc
1506634002.865618	1.000000	21.000000	0.000000	3.000000	1.000000	323.355713	323.355713	0.014136	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634002.866036	1.000000	22.000000	0.000000	3.000000	1.000000	323.355713	323.355713	0.014136	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634002.881793	1.000000	23.000000	0.000000	4.000000	1.000000	323.359039	323.359039	0.034286	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634002.882201	1.000000	24.000000	0.000000	4.000000	1.000000	323.359039	323.359039	0.034286	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634002.895627	1.000000	24.000000	0.000000	4.000000	1.000000	323.359039	323.359039	0.034286	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634002.895627	2.000000	24.000000	0.000000	4.000000	1.000000	323.359039	323.359039	0.034286	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634002.897768	1.000000	25.000000	0.000000	4.000000	1.000000	323.359039	323.359039	0.034286	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634002.913703	1.000000	26.000000	0.000000	4.000000	1.000000	323.359039	323.359039	0.034286	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634002.914060	1.000000	27.000000	0.000000	4.000000	1.000000	323.359039	323.359039	0.034286	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634002.929722	1.000000	28.000000	0.000000	4.000000	1.000000	323.359039	323.359039	0.034286	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634002.934872	8.000000	28.000000	0.000000	4.000000	1.000000	323.359039	323.359039	0.034286	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634002.934872	2.000000	28.000000	0.000000	4.000000	1.000000	323.359039	323.359039	0.034286	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634002.945530	1.000000	29.000000	0.000000	5.000000	1.000000	323.355957	323.355957	0.017700	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634002.945530	1.000000	30.000000	0.000000	5.000000	1.000000	323.355957	323.355957	0.017700	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634002.945530	1.000000	31.000000	0.000000	5.000000	1.000000	323.355957	323.355957	0.017700	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634002.945530	1.000000	32.000000	0.000000	5.000000	1.000000	323.355957	323.355957	0.017700	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634002.952150	1.000000	33.000000	0.000000	5.000000	1.000000	323.355957	323.355957	0.017700	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634002.977617	1.000000	33.000000	0.000000	5.000000	1.000000	323.355957	323.355957	0.017700	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634002.993599	1.000000	34.000000	0.000000	5.000000	1.000000	323.355957	323.355957	0.017700	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634002.993599	1.000000	35.000000	0.000000	5.000000	1.000000	323.355957	323.355957	0.017700	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634002.993599	1.000000	36.000000	0.000000	5.000000	1.000000	323.355957	323.355957	0.017700	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634003.017395	8.000000	36.000000	0.000000	6.000000	1.000000	323.354492	323.354492	-0.032433	1.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634003.017395	1.000000	36.000000	0.000000	6.000000	1.000000	323.354492	323.354492	-0.032433	1.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634003.025490	1.000000	37.000000	0.000000	6.000000	1.000000	323.354492	323.354492	-0.032433	1.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634003.025490	1.000000	38.000000	0.000000	6.000000	1.000000	323.354492	323.354492	-0.032433	1.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634003.041636	1.000000	39.000000	0.000000	6.000000	1.000000	323.354492	323.354492	-0.032433	1.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634003.041970	1.000000	40.000000	0.000000	6.000000	1.000000	323.354492	323.354492	-0.032433	1.000000	0.000000	0.000000	0.000000	1.000000	1.000000	323.336889
1506634003.057593	1.000000	41.000000	0.000000	6.000000	1.000000	323.354492	323.354492	-0.032433	1.000000	0.000000	0.000000	0.000000	2.000000	1.000000	323.351707
1506634003.073964	1.000000	42.000000	0.000000	6.000000	1.000000	323.354492	323.354492	-0.032433	1.000000	0.000000	0.000000	0.000000	2.000000	1.000000	323.351707

Figura 27: File di log

monitorato.

Il codice di *SensorFusionLib* è stato instrumentato con delle *software probe* che includono:

1. La scrittura di un file di log che riporta l'evoluzione dello stato interno del software (figura 27);
2. L'invio verso RTT dei calcoli intermedi compiuti dal software per stimare la posizione del treno. Questi sono:
 - La stima dell'errore commesso sulla predizione delle coordinate geografiche del treno;
 - La stima dell'errore commesso sulla predizione della progressiva chilometrica associata alle coordinate geografiche del treno;
 - La stima della velocità del treno;
 - La stima dell'errore commesso sulla predizione della velocità del treno.

Emergono alcune criticità che è opportuno discutere.

Le *performance* di SFA potrebbero essere degradate dalla *software probe* 1. Dover produrre un file di log implica la necessità di accedere alla memoria di massa, e i tempi di accesso possono essere di diversi ordini di grandezza superiori a quelli richiesti dall'accesso alla memoria centrale (o alla cache) imposto dalla *software probe* 2.

Results of running the Navigation Filter through test data set: "Villa Costanza - Foggini" using the following Edges:

- "Villa Costanza - Foggini" - Length: 4.25044 km

(Body Frame)	Mean Error	Sigma Error	Max Error
X	2.26461 m	1.27891 m	8.01456 m
Y	0.0216197 m	0.0616209 m	0.601101 m
Z	6.21962e-07 m	5.60052e-07 m	3.25963e-06 m
Speed X	0.0346681 ms ⁻¹	0.119348 ms ⁻¹	0.945856 ms ⁻¹
Speed Y	0.0519851 ms ⁻¹	0.142892 ms ⁻¹	1.39831 ms ⁻¹
Speed Z	0.0038464 ms ⁻¹	0.00299897 ms ⁻¹	0.0162497 ms ⁻¹

Details:

- The results have been averaged over 1 simulation
- Duration of test run: 00:07:23
- Software version: 0.0.0
- Time created: 2019-09-22 21:19:19

Figura 28: Report HTML prodotto da RTT

5

ATTIVITÀ Sperimentale

Questo capitolo rappresenta la parte sperimentale del lavoro di Tesi. Nel seguito si illustrano gli esperimenti di *fault injection* condotti sul sistema nell'ambiente descritto nel capitolo 4, e si discutono i risultati ottenuti.

La traccia ferrotramviaria scelta per l'analisi è un sottoinsieme della linea T1 della Tramvia di Firenze, che si estende per 4.25044 chilometri dal terminal di *Villa Costanza*, nel comune di Scandicci.



Figura 29: Traccia di analisi

5.1 MISURE DI INTERESSE

La *fault tolerance* del sistema verrà valutata in termini degli errori che SFA commette nella stima delle seguenti grandezze:

- Posizione del treno, espressa in coordinate ECEF;
- Velocità del treno, proiettata sui tre assi cartesiani.

ECEF è l'acronimo di *Earth Centered Earth Fixed*. Una coordinata ECEF esprime la posizione di un oggetto immerso in un sistema di riferimento cartesiano a 3 dimensioni, con origine nel centro della Terra.

5.2 GUASTI INIETTATI

In relazione al *fault model* individuato al capitolo 4, sono stati iniettati i seguenti guasti:

- (1) Soppressione del canale di comunicazione tra IMU e SFA;
- (2) Alterazione del contenuto dei messaggi trasmessi dai sensori al modulo SFA;
- (3) Soppressione del canale di comunicazione tra Odometro e SFA.

Rispetto al sistema reale descritto nel capitolo 3, i guasti (1) e (3) simulano un guasto hardware nei bus dati che collegano i sensori alla scheda *NVIDIA TX-Jetson* o comunque l'interruzione del funzionamento dei sensori, sia essa permanente o temporanea.

Il guasto (2) simula un comportamento di UDP comunque preventivabile, data la natura *best effort* del protocollo.

Quando si utilizza UDP è opportuno non fare mai assunzioni riguardo il corretto ordinamento e l'integrità dei messaggi trasmessi.

5.3 ESPERIMENTI

Durante l'attività di analisi sono stati effettuati numerosi esperimenti. In questa sezione, si riportano in dettaglio solo i risultati ritenuti più interessanti, mentre i rimanenti vengono descritti a un livello più generico.

La soppressione del canale di comunicazione tra IMU e modulo SFA, quando protratta per un periodo superiore a 5 secondi, ha prodotto un'interruzione del servizio. Questo fatto è in linea con le aspettative in quanto IMU è il sensore principale su cui si basa l'esecuzione di SFA, e senza di esso l'algoritmo non può funzionare.

Il valore limite di 5 secondi trova giustificazione andando a osservare l'implementazione del modulo SFA. Quest'ultimo infatti è programmato in maniera tale che il massimo numero di campionamenti IMU che può

predire attraverso regressione lineare è 500.

La frequenza di campionamento di IMU è pari a:

$$100 \text{ Hz} = 100 \frac{\text{campionamenti}}{\text{s}}$$

Pertanto:

$$\frac{500 \text{ campionamenti}}{100 \frac{\text{campionamenti}}{\text{s}}} = \frac{500}{100} \text{ s} = 5 \text{ s}$$

L'alterazione del contenuto dei messaggi non ha portato a effetti rilevabili, a condizione che il software abbia maturato una adeguata esperienza circa il comportamento corretto dei sensori, ovvero, abbia stimato correttamente i parametri del modello di regressione attraverso il quale vengono predette le misure mancanti.

Nel seguito di questa sezione, si discutono gli esperimenti (3).

Per ciascuna grandezza osservata viene riportato il valore medio, il valore massimo e la deviazione standard (dev. std.) dell'errore commesso da SFA nel relativo processo di stima.

5.3.1 *Golden Run*

Per ottenere un termine di paragone consistente, il sistema viene eseguito prima senza l'iniezione di guasti.

I risultati ottenuti durante la campagna di *fault injection* verranno quindi valutati in relazione ai risultati ottenuti dal software senza introduzione di guasti. Questa prima esecuzione del software prende il nome di *golden run*.

Sensori integrati	Frequenza IMU	Frequenza odometro	Varianza Odometro	Iterazioni
IMU, Odometro	100 Hz	10 Hz	0.0004	10

Tabella 6: Golden Run: workload

Misura	Errore medio	Errore massimo	Dev. std. errore
ECEF X	3.5826 m	20.1141 m	5.60308 m
ECEF Y	0.0243133 m	0.362813 m	0.0452763 m
ECEF Z	3.56432e-06 m	3.19201e-06 m	8.81543e-06 m
Velocità X	0.0169528 m/s	0.124472 m/s	0.0199173 m/s
Velocità Y	0.0394826 m/s	0.847261 m/s	0.0828195 m/s
Velocità Z	0.00382241 m/s	0.0192343 m/s	0.00314704 m/s

Tabella 7: Golden Run: Risultati

5.3.2 Soppressione della comunicazione odometro - SFA

In questo scenario il *faultload* consiste nella soppressione del canale di comunicazione tra Odometro e modulo SFA.

Esperimento 3.1

Si sopprime il canale di comunicazione **per tutta la durata dell'esperimento**.

Sensori integrati	Frequenza IMU	Frequenza odometro	Varianza Odometro	Iterazioni
IMU, Odometro	100 Hz	10 Hz	0.0004	10

Tabella 8: Esperimento 3.1: workload

Misura	Errore medio	Errore massimo	Dev. std. errore
ECEF X	861.883 m	2431.1 m	678.953 m
ECEF Y	348.814 m	1518.65 m	499.222 m
ECEF Z	0.123305 m	0.155086 m	0.567656 m
Velocità X	8.20331 m/s	30.782 m/s	7.32822 m/s
Velocità Y	23.4213 m/s	75.1929 m/s	20.0333 m/s
Velocità Z	87.7399 m/s	87.1907 m/s	245.723 m/s

Tabella 9: Esperimento 3.1: Risultati

Misura	Errore medio	Errore massimo	Dev. std. errore
ECEF X	+23957.5 %	+11986.5 %	+12017.5 %
ECEF Y	+1.43456e+06 %	+4.18477e+05 %	+1.10251e+06 %
ECEF Z	+3.45933e+06 %	+1.77827e+06 %	+1.75916e+06 %
Velocità X	+48289.1 %	+24630.1 %	+36693.2 %
Velocità Y	+59220.6 %	+8774.82 %	+24089.1 %
Velocità Z	+2.29531e+06 %	+1.27743e+06 %	+2.77046e+06 %

Tabella 10: Esperimento 3.1: Confronto con Golden Run

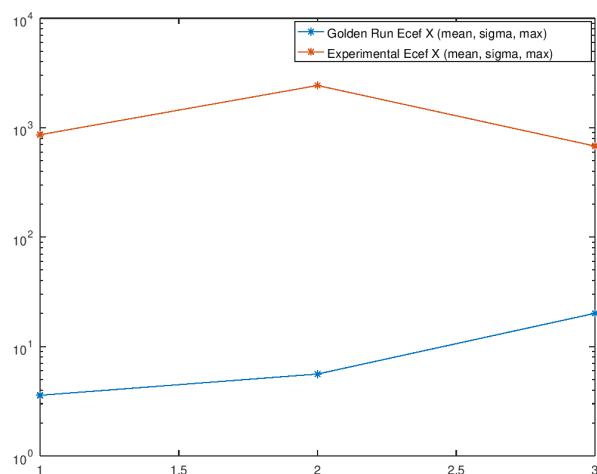


Figura 30: Esperimento 3.1: Grafico di confronto ECEF X con Golden Run

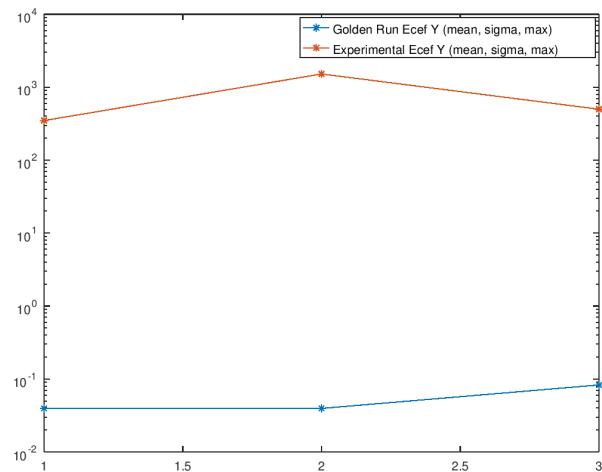


Figura 31: Esperimento 3.1: Grafico di confronto ECEF Y con Golden Run

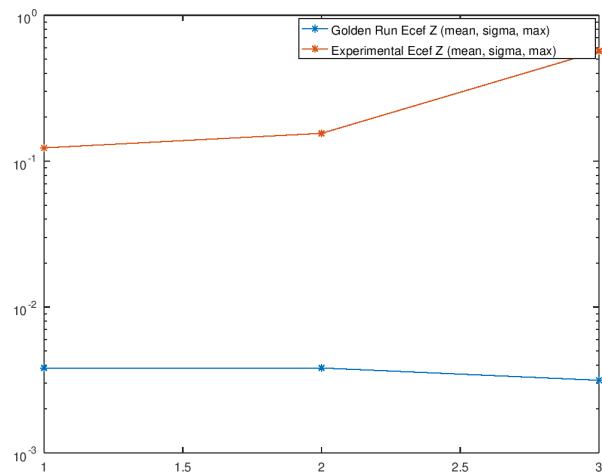


Figura 32: Esperimento 3.1: Grafico di confronto ECEF Z con Golden Run

Alimentare SFA utilizzando esclusivamente le misurazioni di IMU conduce a una netta divergenza dell'errore, sia in termini di posizione che in termini di velocità.

Questi risultati non sono accettabili. In questa fattispecie, tuttavia, è stato iniettato un importante *faultload* altamente improbabile da osservare sul campo.

Esperimento 3.2

Si sopprime il canale di comunicazione tra SFA e Odometro durante la **prima metà della simulazione**.

Sensori integrati	Frequenza IMU	Frequenza odometro	Varianza Odometro	Iterazioni
IMU, Odometro	100 Hz	10 Hz	0.0004	10

Tabella 11: Esperimento 3.2: workload

Misura	Errore medio	Errore massimo	Dev. std. errore
ECEF X	4.3248 m	20.5617 m	5.41018 m
ECEF Y	0.0226056 m	0.39742 m	0.04816 m
ECEF Z	3.81041e-06 m	3.30294e-05 m	9.04865e-06 m
Velocità X	0.0248871 m/s	0.150687 m/s	0.0260141 m/s
Velocità Y	0.0475246 m/s	0.908185 m/s	0.0899591 m/s
Velocità Z	0.00406042 m/s	0.0228906 m/s	0.00340827 m/s

Tabella 12: Esperimento 3.2: Risultati

Misura	Errore medio	Errore massimo	Dev. std. errore
ECEF X	+20.7168 %	+2.22531 %	-3.44275 %
ECEF Y	-7.02373 %	+9.53852 %	+6.36912 %
ECEF Z	+6.90426 %	+3.47524 %	+2.64559 %
Velocità X	+46.8023 %	+21.061 %	+30.6106 %
Velocità Y	+20.3685 %	+7.1907 %	+8.62068 %
Velocità Z	+6.2267 %	+19.0093 %	+8.30082 %

Tabella 13: Esperimento 3.2: Confronto con golden run

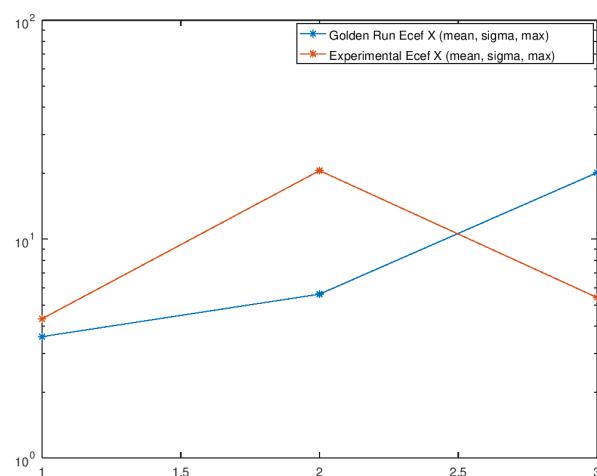


Figura 33: Esperimento 3.2: Grafico di confronto ECEF X con Golden Run

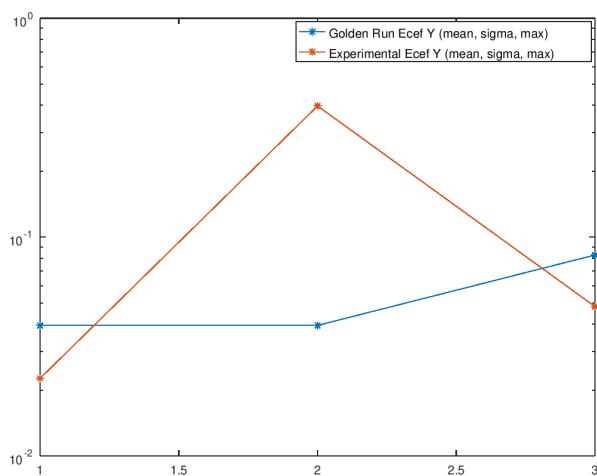


Figura 34: Esperimento 3.2: Grafico di confronto ECEF Y con Golden Run

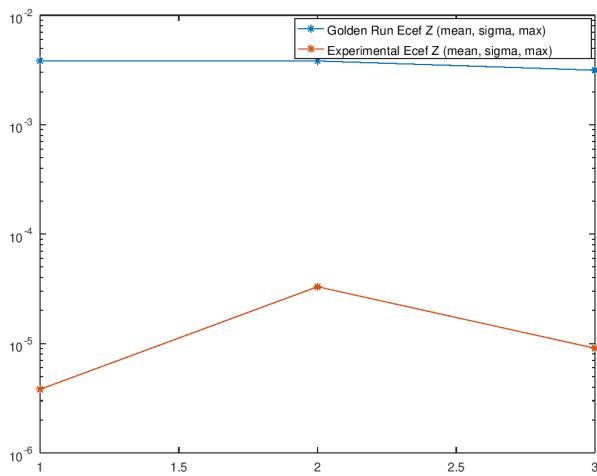


Figura 35: Esperimento 3.2: Grafico di confronto ECEF Z con Golden Run

Si osserva un lieve degrado delle performance che influisce in maniera non particolarmente significativa ai fini del posizionamento.

La grandezza che appare più degradata è la velocità, e questo fatto permane in linea con le aspettative, poiché l'odometro è il sensore preposto a fornire le misure di velocità al sistema.

Esperimento 3.3

Si sopprime il canale di comunicazione tra SFA e Odometro durante **la seconda metà della simulazione**.

Sensori integrati	Frequenza IMU	Frequenza odometro	Varianza Odometro	Iterazioni
IMU, Odometro	100 Hz	10 Hz	0.0004	10

Tabella 14: Esperimento 3.3: workload

Misura	Errore medio	Errore massimo	Dev. std. errore
ECEF X	3.57373 m	20.1609 m	5.60304 m
ECEF Y	0.0234386 m	0.366496 m	0.0445943 m
ECEF Z	3.55578e-06 m	3.19863e-05 m	8.7636e-06 m
Velocità X	0.0184494 m/s	0.129497 m/s	0.0222426 m/s
Velocità Y	0.0396467 m/s	0.863711 m/s	0.084737 m/s
Velocità Z	0.00355928 m/s	0.0189619 m/s	0.00306268 m/s

Tabella 15: Esperimento 3.3: Risultati

Misura	Errore medio	Errore massimo	Dev. std. errore
ECEF X	-0.247586 %	+0.232673 %	-0.000714 %
ECEF Y	-3.59762 %	+1.01512 %	-1.50631 %
ECEF Z	-0.239597 %	+0.207393 %	-0.587946 %
Velocità X	+8.82804 %	+4.03705 %	+11.6748 %
Velocità Y	+0.415626 %	+1.94155 %	+2.31528 %
Velocità Z	-6.88388 %	-1.41622 %	-2.68061 %

Tabella 16: Esperimento 3.3: Confronto con golden run

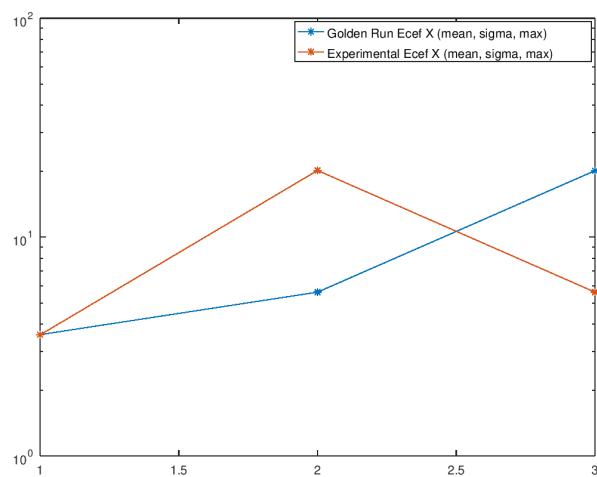


Figura 36: Esperimento 3.3: Grafico di confronto ECEF X con Golden Run

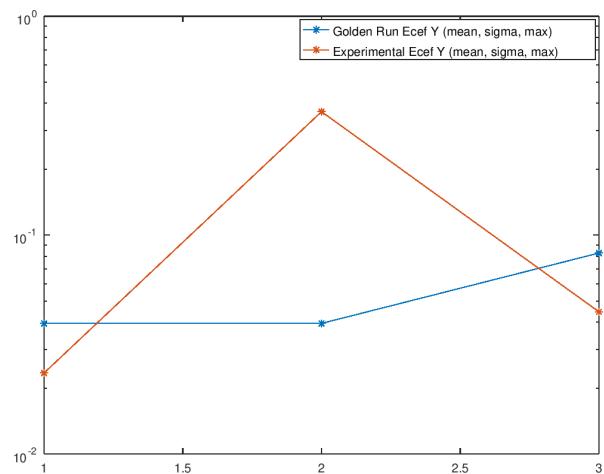


Figura 37: Esperimento 3.3: Grafico di confronto ECEF Y con Golden Run

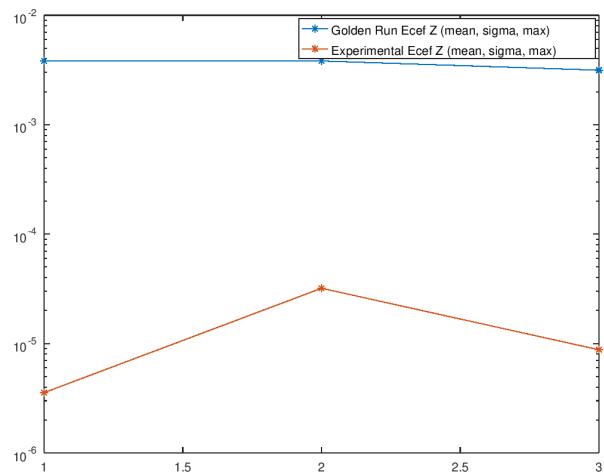


Figura 38: Esperimento 3.3: Grafico di confronto ECEF Z con Golden Run

Non si osservano particolari variazioni rispetto ai risultati ottenuti nella *golden run*. È ragionevole ipotizzare che la causa di questo comportamento sia da ricercare nella sezione di traccia in cui si è soppresso il canale di comunicazione tra Odometro e SFA. Nella prima metà la traccia compie una curva netta, mentre nella seconda essa si sviluppa in modo approssimativamente lineare.

5.4 CONSIDERAZIONI FINALI

La campagna di *fault injection* effettuata sul modulo SFA ha condotto alle conclusioni che seguono:

- Il sistema è in grado di tollerare la perdita di misure IMU fino a un massimo di 5 secondi. Superata questa soglia, il sistema si blocca e smette di erogare il proprio servizio. Questa modalità di fallimento è comunque facilmente rilevabile da OBCU. Quest'ultima potrebbe agire sul treno, ad esempio fermandolo. Si osserverebbe un degrado delle performance, ma si avrebbe comunque un fallimento *safe*.
- Entro certi limiti di approssimazione, il sistema sembra tollerare bene la ricezione di messaggi fuori ordine o alterati. Infatti, attraverso un meccanismo di *regressione lineare*, il sistema è in grado di riconoscere e scartare messaggi che hanno un'elevata probabilità di contenere valori non affidabili. Un messaggio ricevuto fuori ordine è semplicemente un messaggio che ha probabilità 1 di non essere affidabile.

Quando un messaggio viene scartato, l'informazione mancante viene stimata e fornita al sistema come se fosse stata campionata dai sensori.

- L'odometro sembra particolarmente utile quando il movimento del treno è caratterizzato da repentina cambi di direzione, come nel caso di curve molto strette. Su tratti sufficientemente lineari, il sistema sembra comportarsi correttamente anche se viene alimentato esclusivamente dal sensore inerziale.
- Non è stato possibile osservare il sistema quando sottoposto a misure di tipo GPS, e le *software probe* che sono state inserite all'interno del codice potrebbero aver degradato le performance, seppur trascurabilmente. Si ipotizza che l'aggiunta di un nuovo strumento di misura, come il GPS, possa migliorare le performance osservate.

- Quando la stima della posizione non diverge, è stato osservato che l'errore massimo sulla stima della posizione permane, in modulo, pari a circa 20 metri.

6

CONCLUSIONI

In questa Tesi, si è discusso un processo di analisi sperimentale condotto verso un sottosistema di posizionamento ferrotramviario.

L'analisi condotta rientra nella categoria *fault injection*, la quale prevede l'osservazione diretta del sistema, o di un suo prototipo, nel suo reale ambiente di esecuzione. Il fine di un'attività di *fault injection* è quello di raccogliere accurate misure sperimentali circa la *dependability* del sistema analizzato, quando vengono inseriti volontariamente dei guasti al suo interno. Si è discusso il concetto di *dependability*, definendola come la capacità che ha un sistema di fornire un servizio in modo corretto. In particolare, sono stati individuati:

- Le *measures*, ovvero i parametri di valutazione della *dependability*;
- I *threats*, ossia gli eventi che minano la *dependability* di un sistema;
- I *means*, insieme di tecniche e metodologie atte a raggiungere la *dependability* di un sistema informatico.

La *dependability* è un aspetto fondamentale per tutti i sistemi informatici, ma il suo raggiungimento diventa obbligatorio per sistemi operanti in contesti *safety critical*, come ad esempio il settore ferroviario.

Sono stati discussi gli standard EN 50126 e EN50128 che regolamentano la gestione e il raggiungimento della *safety* nei sistemi ferroviari, e le normative operazionali imposte dallo standard europeo ERTMS/ETCS, completando il quadro del contesto normativo e operativo in cui si colloca il sistema studiato.

Il sistema è stato poi classificato, da un punto di vista architetturale, come un *Cyber Physical System of Systems*, quindi se ne sono descritti gli elementi chiave, come le interfacce e i sistemi costituenti. La definizione delle interfacce è particolarmente importante poichè qualunque struttura del sistema, responsabile del comportamento osservato, può essere ridotta

alla specifica delle interfacce del sistema.

La descrizione del sistema si è quindi conclusa con la specifica dei software che lo compongono.

Definito il sistema nominale, la discussione si è concentrata sulla descrizione dell'ambiente di analisi costruito al fine di garantire le proprietà che un sistema di *monitoring* e *fault injection* deve possedere:

- Non intrusività;
- Rappresentatività;
- Ripetibilità;
- Fattibilità.

Sono stati quindi descritti gli esperimenti condotti individuando:

- Un *fault model* rappresentativo per il sistema;
- Le misure di interesse che si intende valutare;
- Un *workload* il più possibile conforme agli input reali che il sistema dovrà processare;
- Un *faultload* basato sui requisiti di sistema e sulle tecnologie impiegate.

Attraverso l'attività di *fault injection* è stato principalmente osservato che:

1. Il sistema è in grado di tollerare bene guasti al canale di comunicazione verso i sensori, a condizione che IMU sia sempre funzionante e che la traccia sia sufficientemente lineare;
2. Il sistema è capace di individuare messaggi che hanno un'elevata probabilità di essere errati, e quindi di correggerne il contenuto;
3. Se IMU non fornisce messaggi al sistema per più di due secondi, questo termina la sua esecuzione in una maniera comunque rilevabile facilmente da altri meccanismi di controllo installati a bordo.

I risultati che ha fornito la campagna di *fault injection* sembrano promettenti nell'ottica di poter impiegare il sistema sul campo, sebbene questo sia ancora in fase di sviluppo e si è reso disponibile per l'analisi solamente un suo prototipo.

BIBLIOGRAFIA

- [1] J.C. Laprie, *Dependability - its attributes impairments and means, Predictability Dependable Computing Systems*, Springer (1995) (Cited on page 9.)
- [2] J.C. Knight, *Safety Critical Systems: Challenges and Directions, Proceedings of the 24th International Conference on Software Engineering* (2002) (Cited on page 11.)
- [3] A. Bondavalli, *L'analisi quantitativa dei Sistemi Critici, Fondamenti e Tecniche per la Valutazione - Analitica e Sperimentale - di Infrastrutture Critiche e Sistemi Affidabili* (2011) (Cited on page 12.)
- [4] G.J. Nutt, *Tutorial: Computer system monitors*, Computer (1975) (Cited on page 13.)
- [5] B. Plattner, *Real-time execution monitoring*. *IEEE Transactions on Software Engineering* (1984) (Cited on page 13.)
- [6] B. Plattner, J. Nievergelt, *Special feature: Monitoring program execution: A survey*, Computer (1981) (Cited on page 13.)
- [7] M. Vieira, *Fault Injection and Robustness Testing - Enabling Techniques for Assessing Computer Systems*, University of Coimbra, Portugal (Cited on page 15.)
- [8] M. Vieira, *Assessing the robustness and security of Web Services*, AMBER, University of Coimbra, Portugal (Cited on page 15.)
- [9] H. Madeira, *Design for experiments for resilience assessment and benchmarking*, University of Coimbra, Portugal (Cited on page 15.)
- [10] K. Wolter et al, *Resilience Assessment and Evaluation of Computing Systems*, Springer (2012) (Cited on page 16.)
- [11] CENELEC European Committee for Electrotechnical Standardization, *EN 50126:2000 - Railway applications - The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS)* (2000) (Cited on pages 12 and 20.)

- [12] CENELEC European Committee for Electrotechnical Standardization. *EN 50128:2011 - Railway Applications - Communications, signalling and processing systems - Software for railway control and protection systems* (2011) (Cited on page 21.)
- [13] International Electrotechnical Commission, *61508-1: Functional safety of electrical/electronic/programmable electronic safety-related systems, edition 2.0* (2010) (Cited on page 21.)
- [14] J. Otegui, *A Survey of Train Positioning Solutions*, *IEEE Sensors Journal*, Vol. 17, No. 20 (2017) (Cited on page 21.)
- [15] T. Albrecht et al, *A precise and reliable train positioning system and its use for automation of train operation*, *Proc. IEEE Int. Conf. Intell. Rail Transp.* (2013) (Cited on page 22.)
- [16] European Commission, *Delivering an effective and interoperable European Rail Traffic Management System (ERTMS) - the way ahead* (2017) (Cited on page 22.)
- [17] A. Neri, F. Rispoli, P. Salvatori, *An analytical assessment of a GNSS-based train integrity solution in typical ERTMS level 3 scenarios*, in *Proc. Eur. Navigat. Conf. (ENC)*, Bordeaux, France, (2015) (Cited on page 22.)
- [18] P. Josserand, F.H Willard, *Rights of Trains* (5th ed.), Simmons-Boardman Publishing Corporation, New York (1957) (Cited on page 22.)
- [19] N.A. Zafar et al, *Towards the safety properties of moving block railway interlocking system*, *International Journal of Innovative Computing Information and Control* (2012) (Cited on page 23.)
- [20] R. S. Hosse, H. Manz, K. Burmeister, E. Schnieder, *Market analysis for satellite train localisation for train control systems*, *Proc. 5th Conf. Transp. Solutions Res. Deployment Transp.* (2014) (Cited on page 23.)
- [21] J. Marais, J. Beugin, M. Berbineau, *A Survey of GNSS-Based Research and Developments for the European Railway Signaling*, *IEEE transactions on intelligent transportation system* (2017) (Cited on page 23.)
- [22] A. Mirabadi et al, *Application of sensor fusion to railway systems*, *IEEE* (1996) (Cited on page 24.)
- [23] A. Ceccarelli et al, *Basic Concepts on Systems of Systems, Cyber-Physical Systems of Systems*, Springer (2017) (Cited on page 25.)

- [24] F. Bohringer, A. Geistler, *Adaptation of the kinematic train model using the interacting multiple model estimator*, *Advances in Transport*, vol. 74, no. 7. Southampton (2004) (Cited on page 29.)
- [25] B. Cai, X. Wang, *Train positioning via integration and fusion of GPS and inertial sensors*, *WIT Transactions on the Built Environment*, Southampton (2000) (Cited on page 30.)
- [26] M. Malvezzi et al, *A localization algorithm for railway vehicles based on sensor fusion between tachometers and inertial measurement units*, *Proc. Inst. Mech.* (Cited on page 30.)
- [27] C. Reimer et al, *INS/GNSS/odometer data fusion in railway applications*, *Proc. DGON Intertial Sensors Syst. (ISS)*, vol. 2. Karlsruhe, Germany (2016) (Cited on page 30.)
- [28] L. Junyan et al, *Application Research of Embedded Database SQLite*, IEEE (2009) (Cited on page 30.)
- [29] X. Liu, A. Goldsmith, *Kalman Filtering with Partial Observation Losses*, Department of Electrical Engineering, Stanford University, Stanford, USA (Cited on page 31.)
- [30] R. Mazl, L. Preucil, *Sensor Data Fusion for Inertial Navigation of Trains in GPS Dark Areas (Mathematics in Science and Engineering)*, San Diego, CA, USA (2003) (Cited on pages 31 and 43.)
- [31] S. Mittal, *A Survey on optimized implementation of deep learning models on the NVIDIA Jetson platform*, *Journal of Systems Architecture Volume 97* (2019) (Cited on page 32.)
- [32] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications 2nd edn*, Springer, New York (2011) (Cited on page 26.)
- [33] H. Kopetz, W. Ochsenreiter, *Clock synchronization in distributed real-time systems*, IEEE (1987) (Cited on page 38.)
- [34] M.J. Pont, *Patterns for Time-Triggered Embedded Systems*, Addison-Wesley (2001) (Cited on page 35.)
- [35] H. Kopetz, *Event-Triggered versus Time-Triggered Real-Time Systems*, Springer (1991) (Cited on page 35.)
- [36] S. Dilhaire, D. Maillet, *Dealing with the measurement noise of a sensor* (2015) (Cited on page 42.)