

AeroGuard - MAS

Multi-Agent Airspace Conflict Manager

Indice

- Contesto Aeronautico.
- Separazione e prevenzione di conflitti.
- TCAS e incidenti reali di riferimento.
- Bias e fattore umano.
- Obiettivi del progetto.
- Architettura.
- Intelligenza: Prolog, agenti jason, STRIPS
- GUI, Test, CI/CD, Conclusioni.

Indice

- Contesto Aeronautico.
- Separazione e prevenzione di conflitti.
- TCAS e incidenti reali di riferimento.
- Bias e fattore umano.
- Obiettivi del progetto.
- Architettura.
- Intelligenza: Prolog, agenti jason, STRIPS
- GUI, Test, CI/CD, Conclusioni.

Perché i conflitti aerei sono critici?

Gli aereomobili condividono uno spazio tridimensionale caratterizzato da:

- Posizione orizzontale
- Quota
- Tempo



Un conflitto nasce quando due traiettorie portano gli aerei troppo vicini nello stesso intervallo temporale.

Aerodinamica

Un aereo vola grazie all'equilibrio tra quattro forze

- Portanza
- Peso
- Spinta
- Resistenza

Nel progetto non simulo la fisica reale del volo, ma astraggo l'aereo come entità che si muove tra waypoint con quota e velocità semplificate.

Livelli di astrazione

Reale	AeroGuard-MAS
Coordinate Geografiche	Coordinate 2D
Altitudine / Flight Level	FlightLevel
Velocità reale	Velocità per tick
Piano di volo	Route
Navigazione	WayPoint
Tempo continuo	Tick discreti
ATC / TCAS	Agenti + Reasoning + Planning

AeroGuard-Mas Separazione

Nel progetto un conflitto avviene quando:

```
1 distanza orizzontale < soglia orizzontale  
2 e  
3 distanza verticale < soglia verticale
```

Esempio:

```
1 horizontal = 5.0  
2 vertical = 1000 ft
```

Conflitto corrente e Conflitto predetto

Conflitto corrente: la separazione è già sotto soglia.

Conflitto predetto: la separazione sarà sotto soglia entro alcuni tick futuri.

Il sistema non aspetta la collisione: cerca di anticipare il problema

TCAS - (Traffic Collision Avoidance System)

TCAS è un sistema di bordo per evitare collisioni

In modo semplificato:

- Osserva altri aeromobili tramite transponder
- Valuta il rischio di collisione in maniera algoritmica
- Avvisa l'equipaggio con manovre correttive diverse per aereo
- Adatta la correzione in base all'opzione scelta dai piloti

TCAS - (Traffic Collision Avoidance System)

<https://www.youtube.com/embed/R5sxW6lscVM?enablejsapi=1>

Traffic Advisory e Resolution Advisory

TA - Traffic Advisory

Segnala il traffico potenzialmente pericoloso

RA - Resolution Advisory

Suggerisce una manovra per evitare il conflitto

In AeroGuard-MAS riprende questo schema

osserva → predice → valuta → decide → spiega

TCAS & AeroGuard-MAS

TCAS	AeroGuard-MAS
Sistema avionico reale	Simulazione didattica
Safety-Critical	Non Safety-Critical
Logica certificata	Regole semplificate
A bordo dell'aereo	Architettura multi-agente
Ambiente reale	Modello discreto 2D + quota

Scenari reali

Gli scenari sono ispirati a pattern reali:

- Überlingen 2002;
- Charkhi Dadri 1996;
- Zagreb 1976;
- Grand Canyon 1956;
- Cerritos 1986.

Sono stati ricostruiti in base alle perizie post-incidente

Scenari reali - Astrazione

Pattern ricorrenti:

- Rotte convergenti
- Quote non mantenute
- Errori procedurali
- Conflitti tra istruzioni
- Perdita di situation awarness
- Difficoltà a reagire in poco tempo

BIAS e Fattore umano

Alcuni fattori che possono portare a errore:

- Confirmation BIAS
- Expectation BIAS
- Automation BIAS
- Plan Continuation BIAS
- Startle Effect
- Situation Awareness

BIAS e Fattore umano

AeroGuard-MAS cerca di ridurre alcuni bias attraverso:

- Regole esplicite
- Decisioni tracciate
- Spiegazioni
- Logging strutturato
- Visualizzazioni della timeline
- Separazione tra osservazione, decisione e spiegazione

Obiettivo del progetto

Realizzare un sistema multi-agente capace di:

- Caricare scenari
- Simulare aeromobili
- Rilevare conflitti
- Ragionare su vincoli di sicurezza
- Pianificare manovre
- Applicare decisioni
- Spiegare il risultato
- Visualizzare il comportamento

Intelligenza

Nel progetto **intelligenza** significa:

- Percezione dello stato
- Detection di situazioni critiche
- Reasoning simbolico
- Pianificazione
- Autonomia decisionale
- Controllo delle conseguenze
- Spiegabilità

Autonomia

Il sistema è **autonomo** perché:

- Osserva lo stato simulato
- Rileva un conflitto
- Sceglie quale aereo manovrare
- Genera una manovra
- Scarta alternative non sicure
- Applica decisioni
- Produce spiegazione

Scenari Principali

Scenari demo:

- simple_conflict.json
- emergency_priority.json
- secondary_conflict.json
- weather_replanning.json
- no_conflict.json

Scenari di stress:

- weather multi-waypoint
- massive convergent airplane

Flow

JSON scenario

- Scenario loader
- SimulationState
- ConflictDetector
- SafetyReasoner Prolog
- STRIPS planner
- ManeuverApplier
- JSONL events
- Streamlit GUI

Separazione dei concetti

Kotlin core:

- Simula
- Ragiona
- Pianifica
- Applica manovre
- Genera eventi

GUI Python:

- Legge JSONL
- Visualizza
- Decide

Interfacce sulle integrazioni complesse

```
1 interface SafetyReasoner {  
2     fun isConflictUnsafe(...)  
3     fun isManeuverAllowed(...)  
4     fun priorityOf(...)  
5     fun explainDecision(...)  
6 }
```

Isolo prolog dal resto del sistema

JSONL Come Osservabilità

```
1 {"tick":4,"type":"plan_generated","aircraft":"DLH456","actions":["climb(DLH456,32000)"]}
```

Ogni linea del file rappresenta un evento

Dove si trova l'intelligenza?

L'intelligenza non è in un singolo punto, è distribuita tra:

- Agenti Jason
- Regole Prolog
- STRIPS planner
- Simulazione predittiva
- Secondary conflict prevention
- Explanation Layer

Prolog

Prolog contiene conoscenza dichiarativa

- Questo conflitto è unsafe?
- Questa manovra è ammessa?
- Quale aircraft ha priorità?
- Quale fatto spiega la decisione?

Prolog - Fatti

```
1 aircraft(aza123).  
2 altitude(aza123, 30000).  
3 priority(aza123, normal).  
4 emergency(aza123, false).  
5  
6 horizontal_distance(aza123, dlh456, 3.2).  
7 vertical_distance(aza123, dlh456, 0).  
8  
9 min_horizontal_separation(5.0).  
10 min_vertical_separation(1000).
```

Questi fatti derivano dallo stato di Kotlin

Prolog - Regole

```
1 unsafe_pair(A1, A2) :-  
2     horizontal_distance(A1, A2, H),  
3     vertical_distance(A1, A2, V),  
4     min_horizontal_separation(MH),  
5     min_vertical_separation(MV),  
6     H < MH,  
7     V < MV.
```

La regola combina dimensione orizzontale e verticale.

Prolog - Regole

```
1 priority_score(Aircraft, 100) :-  
2     emergency(Aircraft, true).  
3  
4 priority_score(Aircraft, 80) :-  
5     low_fuel(Aircraft, true).  
6  
7 priority_score(Aircraft, 10) :-  
8     priority(Aircraft, normal).
```

La priorità viene calcolata simbolicamente, non hard-coded nella simulazione.

Prolog - Regole

```
1 maneuver_allowed(Aircraft, climb(TargetAltitude)) :-  
2     altitude(Aircraft, CurrentAltitude),  
3     valid_altitude_change(CurrentAltitude, TargetAltitude),  
4     TargetAltitude =< 40000.
```

Il planner propone, ma il reasoner valida.

Agenti Jason

Gli agenti rappresentano ruoli autonomi

Nel progetto ci sono:

Agente	Responsabilità
Aircraft	pubblica stato aircraft
Sector	coordina il settore
Conflict Detector	rileva conflitti
Resolution Planner	delega o genera risoluzioni
Explanation Agent	produce spiegazioni

Ogni agente ha responsabilità distinta

Agenti Jason - Beliefs, Goals, Intentions

Nel modello BDI:

- Beliefs: ciò che l'agente sa
- Goals: ciò che l'agente vuole ottenere
- Intentions: piani attivi per raggiungere obiettivi

Esempio:

```
1 agent_role(sector_controller).  
2 sector(sector_alpha).  
3  
4 !coordinate_sector.  
5 !request_conflict_scan.
```

Agenti Jason - Comunicazione Tra Agenti

Il controller delega la resolution al planner

Esempio:

```
1 +conflict_detected(A1, A2, Tick)
2   <- .send(resolution_planner,
3           achieve,
4           resolve_conflict(A1, A2, Tick)).
```

STRIPS Planning - Perché?

La detection dice -> C'è un problema

Il planning risponde -> Quale sequenza di azioni porta a uno stato sicuro?

STRIPS usa:

- Stato iniziale
- Goal
- Azioni
- Precondizioni
- Effetti positivi
- Effetti negativi

STRIPS Planning - Struttura

Ogni azione viene descritta in modo dichiarativo

```
1 data class StripsAction(  
2     val name: String,  
3     val preconditions: Set<Proposition>,  
4     val addEffects: Set<Proposition>,  
5     val deleteEffects: Set<Proposition>,  
6     val maneuver: Maneuver?  
7 )
```

Il problema viene costruito a partire dal conflitto rilevato

```
1 StripsProblem(  
2     initialState = setOf(  
3         conflictActive(conflictId)  
4     ),  
5     goal = setOf(  
6         conflictResolved(conflictId)  
7     ),  
8     actions = candidateActions  
9 )
```

Ricerca del planner

Il planner usa una ricerca BFS limitata

inizializza coda

prendi stato

se goal soddisfatto → ritorna piano

per ogni azione applicabile: applica effetti & aggiungi nuovo stato

Manovre candidate

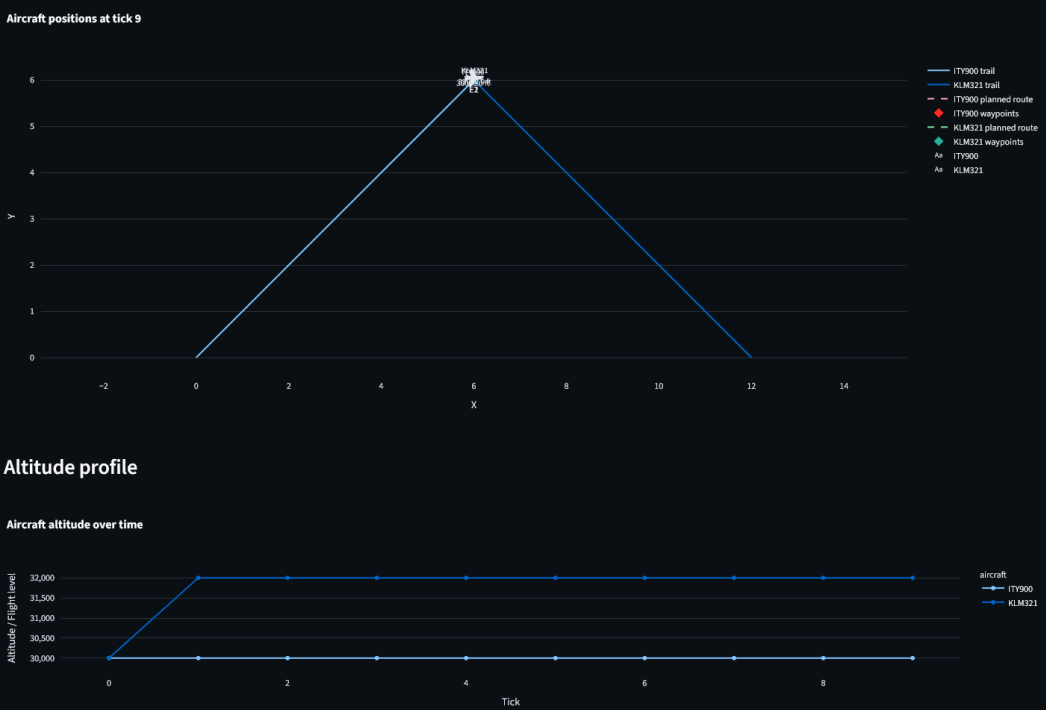
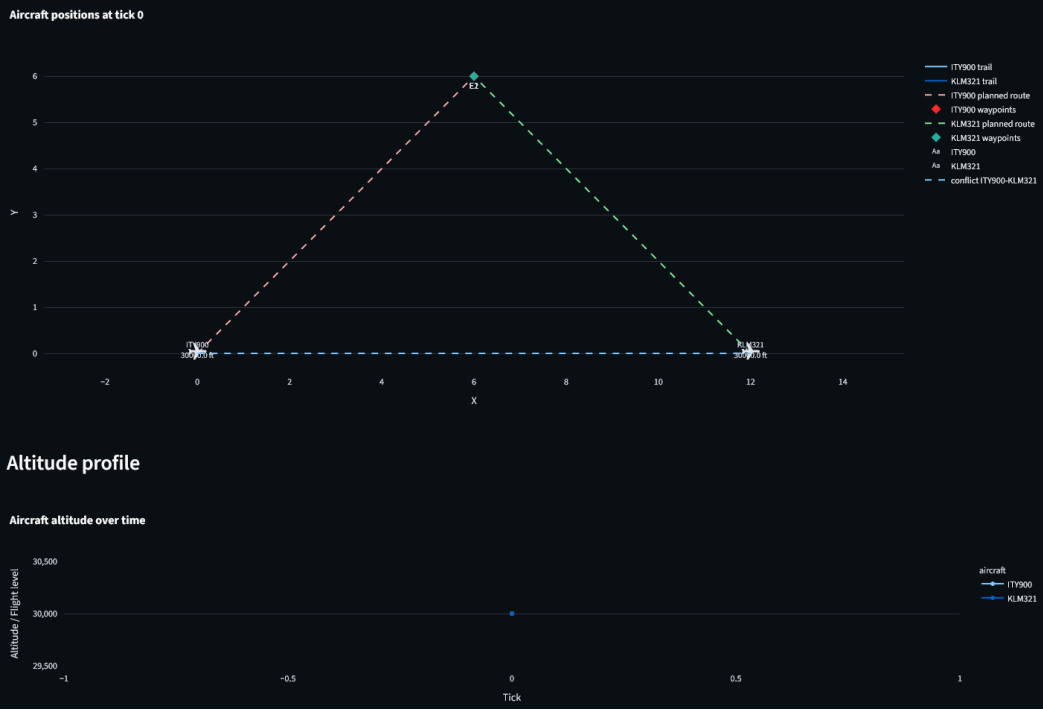
La scelta dell'aereo da manovrare dipende anche dalla priorità prolog

```
1 climb(A, current + 2000)
2 descend(A, current - 2000)
3 slow_down(A)
4 reroute_to_waypoint(A, WP)
```

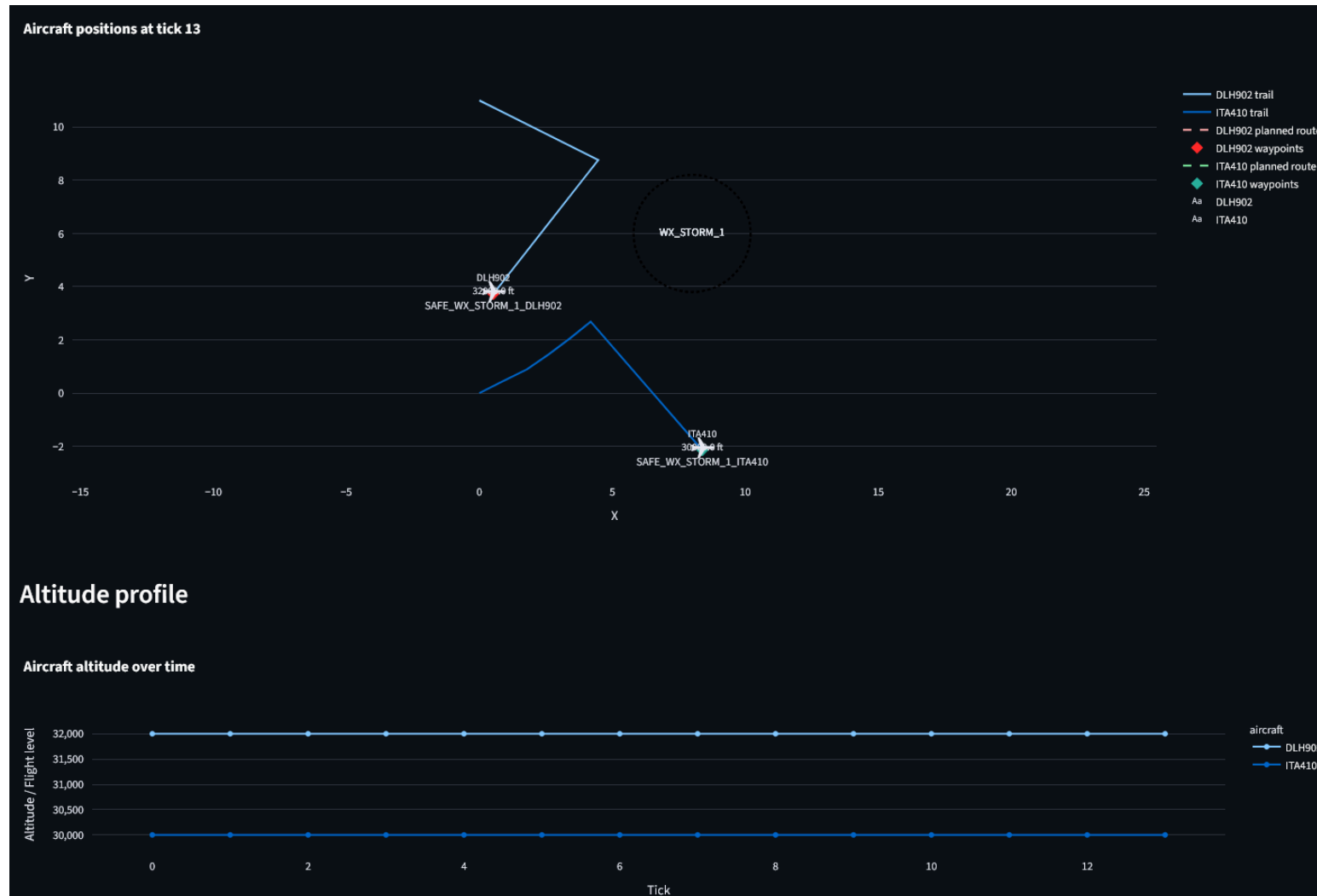
Conflitto Secondario e Ripianificazione Meteo

Il sistema è in grado anche di evitare un conflitto secondario e di ripianificare la rotta partendo da una condizione meteo sfavorevole

GUI



GUI



GRAZIE PER L'ATTENZIONE