# **Pianificazione Automatica per Operazioni di un Team MotoGP**

Il progetto “Pianificazione Automatica per Operazioni di un Team MotoGP” mira a definire in modo automatico la sequenza di operazioni necessarie per preparare le moto da gara partendo da uno stato iniziale prestabilito. In altre parole, il sistema deve pianificare le attività per passare dallo stato “non pronta” allo stato “pronta e testata” delle moto. Per fare questo si utilizzano strumenti di *automated planning*: in particolare il planner ENHSP (Expressive Numeric Heuristic Search Planner) che legge file PDDL di dominio e problema, e produce come output un piano (sequenza ordinata di azioni). I domini e i problemi sono descritti in PDDL (Planning Domain Definition Language), mentre una dashboard grafica realizzata in Python con Pygame visualizza in tempo reale le azioni eseguite sulle moto man mano che il piano viene eseguito.

### **Dominio e Problema in PDDL**

In PDDL la definizione di dominio descrive gli aspetti “universali” del problema: i tipi di oggetti, i predicati, le azioni possibili. Nel mio progetto, il dominio *motogp\_domain.pddl* definisce il tipo moto, i predicati che rappresentano lo stato di ciascuna moto: benzina, elettronica e assetto, e le azioni elementari che possono essere eseguite:

(:predicates

(benzina-ok ?m - moto)

(elettronica-ok ?m - moto)

(assetto-ok ?m - moto)

(pronto ?m - moto)

(tested ?m - moto)

)

(:action benzina

:parameters (?m - moto)

:precondition ()

:effect (benzina-ok ?m)

)

…

(:action test-moto

:parameters (?m - moto)

:precondition (pronto ?m)

:effect (tested ?m)

)

Il problema PDDL descrive invece l’istanza specifica da risolvere: elenca gli oggetti presenti nel mondo, lo stato iniziale (quali predicati sono veri) e lo stato obiettivo desiderato. Ad esempio, nel file problem.pddl con tre moto vengono dichiarati tre oggetti moto1, moto2, moto3 di tipo moto, si assume inizialmente che nessuna moto abbia condizioni “ok”, e si specifica l’obiettivo finale. Un possibile obiettivo è raggiungere (elettronica-ok moto1), (benzina-ok moto2) e (tested moto3). Nel nostro file di esempio il goal è:

(:objects moto1 moto2 moto3 - moto)

(:init

;; (nessuna condizione vera all'inizio)

)

(:goal (and

(elettronica-ok moto1)

(benzina-ok moto2)

(tested moto3)

))

Questo significa che il planner dovrà generare le azioni necessarie affinché, partendo dalle moto non pronte, si raggiunga il goal.

### **Precondizioni e Azioni**

Nel dominio presentato, *la maggior parte delle azioni non ha precondizioni* (ossia può essere eseguita in qualsiasi momento su una moto). In particolare le azioni benzina, assetto ed elettronica sono indipendenti e impongono i rispettivi predicati “ok” senza richiedere nulla. Le azioni con precondizioni sono invece quelle di ordine finale. In PDDL le precondizioni di un’azione rappresentano i predicati che devono essere veri prima di poter eseguire quell’azione. Ad esempio, l’azione l’azione da-testare richiede che (benzina-ok ?m), (elettronica-ok ?m) e (assetto-ok ?m) siano tutti veri, quindi la moto deve aver eseguito tutti i controlli che sono stati indicati dall’utente prima di poterla considerare “pronta”. L’azione test-moto richiede (pronto ?m) come precondizione e produce (tested ?m) come effetto. Questo perché ho pensato che se una moto necessita di regolazione elettronica, regolazione assetto e rifornimento benzina allora solo in questo caso deve essere testa. Le precondizioni sono dunque il vincolo che deve essere soddisfatto nel piano di azioni: il planner applicherà ogni azione solo quando le sue precondizioni risultano vere nello stato corrente.

### **Azioni del Dominio e Problemi Specifici**

Sintetizzando, il nostro dominio *motogp\_domain* include le seguenti azioni principali:

* benzina(?m): rifornisce la moto ?m. Effetto: (benzina-ok ?m). (Nessuna precondizione)
* assetto(?m): regola l’assetto della moto ?m. Effetto: (assetto-ok ?m). (Nessuna precondizione)
* elettronica(?m): verifica l’elettronica di ?m. Effetto: (elettronica-ok ?m). (Nessuna precondizione)
* da-testare(?m): azione intermedia che trasforma una moto in “pronta”. Precondizione: (and (benzina-ok ?m) (assetto-ok ?m) (elettronica-ok ?m)); effetto: (pronto ?m).
* test-moto(?m): esegue il test finale sulla moto ?m. Precondizione: (pronto ?m); effetto: (tested ?m).

Per quanto riguarda i problemi, oltre all’esempio a tre moto si può considerare anche un caso più semplice con una sola moto. Nel caso di 1 moto, il file problema avrebbe un solo oggetto moto1 e un goal adeguato (ad es. (benzina-ok moto1) se si vuole testare il corretto rifornimento). Il file problema con 3 moto mostrato sopra prevede invece tutti e tre gli oggetti moto1, moto2, moto3 non inizialmente preparati e un goal costituito come nell’esempio:

(:objects moto1 moto2 moto3 - moto)

(:goal (and

(elettronica-ok moto1)

(benzina-ok moto2)

(tested moto3)

))

Nel complesso, il dominio definisce cosa è possibile fare in generale (le azioni), mentre ogni problema concreto specifica su quali oggetti (moto) si opera e quali condizioni iniziali e finali interessano.

## **Funzionamento del Planner (ENHSP)**

Per generare il piano operativo, si utilizza il planner ENHSP: si forniscono in input al sistema i file domain.pddl e problem.pddl descritti. ENHSP trasforma la pianificazione in un problema di ricerca e restituisce come risultato un piano (ossia una sequenza di azioni) che conduce dallo stato iniziale allo stato obiettivo. In pratica il planner esplora lo spazio degli stati applicando le azioni finché non vengono soddisfatte le condizioni dell’obiettivo.

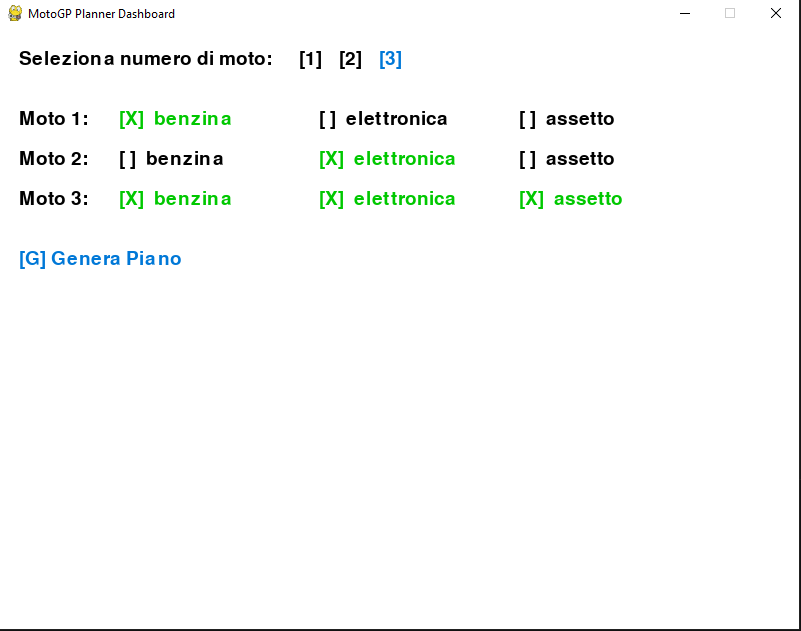
Durante la ricerca, ENHSP (che è un planner euristico) utilizza funzioni di costo e stima per guidare l’espansione. In un tipico approccio di ricerca A\*, a ogni nodo (stato) sono associati due valori chiave: g(n) è il costo del percorso dall’inizio al nodo n, e h(n) è una stima euristica del costo restante fino all’obiettivo. Il costo totale stimato è g(n) + h(n). Il planner prova ad espandere i nodi a basso valore di f(n), avvicinandosi così all’obiettivo in maniera guidata. Va poi distinto il caso di pianificazione valida da quello di pianificazione ottima. Nel primo caso basta trovare un qualsiasi piano che soddisfi l’obiettivo, mentre nel secondo si ricerca un piano di costo minimo. In questo progetto base ci prefiggiamo di ottenere un piano valido; l’ottimalità (ad es. minimizzare il numero di azioni totali) è un possibile sviluppo futuro.

## **Dashboard Python**

Una volta calcolato il piano con ENHSP, esso viene letto da un programma Python che aggiorna la visualizzazione grafica in tempo reale. La dashboard è implementata con la libreria Pygame. Essa offre un’unica superficie di visualizzazione e le modifiche non sono visibili fino a quando non si invoca la funzione di aggiornamento (flip). Nel nostro software questo significa che, ad ogni passo del piano, aggiorniamo i parametri dello stato di ciascuna moto (ad es. colore o immagine che segnala “pronta” o “testata”) e poi chiamiamo pygame.display.flip() per renderli visibili.



*Figura: Esempio di dashboard grafica con una sola moto. Man mano che il piano si esegue, lo stato della moto viene aggiornato.*



*Figura: Esempio di dashboard con tre moto.* In pratica, all’inizio la dashboard visualizza tutte le moto come “non pronte”. Ad ogni passo del piano si esegue la corrispondente azione sul modello interno (ad es. assetto moto3 imposta il predicato (assetto-ok moto3)). Alla fine dell’esecuzione, tutte le condizioni richieste nel goal saranno soddisfatte e le moto risultano “pronte” o “testate” nell’interfaccia.

## **Conclusioni**

Il progetto realizzato dimostra la fattibilità di automatizzare la preparazione delle moto di un team MotoGP tramite la pianificazione automatica: abbiamo ottenuto piani validi che portano ogni moto allo stato desiderato, e una visualizzazione intuitiva che mostra in tempo reale lo stato di avanzamento.

Tra gli sviluppi futuri si potrebbe aggiungere il supporto alla pianificazione parallela (per eseguire azioni contemporaneamente su moto diverse) o anche l’integrazione di dati reali provenienti dai sensori delle moto (ad es. livello di carburante, stato delle centraline) per aggiornare dinamicamente il dominio, e gestire problemi più complessi.

Tommaso Garzetti