NavigAItors

Progetto di Robotica

Università degli Studi di Palermo

Ingegneria Informatica | LM-32

Agostino Messina
Alberto Conti
Federico Princiotta Cariddi

Descrizione del progetto

Il progetto consiste in una simulazione ambientata in un campo di 150x150 celle, dove un drone e un robot terrestre collaborano per navigare un ambiente ostile. Il drone, sorvolando l'area, identifica e segnala la posizione di nemici e ostacoli al robot. Quest'ultimo, equipaggiato con un algoritmo di pathfinding A*, calcola e aggiorna continuamente il percorso più sicuro ed efficiente per raggiungere un obiettivo predefinito. La simulazione si svolge all'interno dell'engine Unity, e fa uso di una macchina a stati per gestire i diversi comportamenti e stati operativi del robot, inclusi la pianificazione del percorso, la navigazione e l'arrivo a destinazione. L'obiettivo è sviluppare un sistema dinamico che possa adattarsi in tempo reale a cambiamenti dell'ambiente, testando l'efficacia di strategie di intelligenza artificiale in scenari complessi di navigazione autonoma.

Ambiente di sviluppo

Il progetto è stato sviluppato utilizzando il Game Engine Unity, mediante il quale è stato possibile costruire l’intero ambiente di simulazione, le relazioni tra i GameObject e, mediante una serie di script in C#, l’intera logica dell’applicazione. Unity offre una facile gestione delle associazioni tra scripts e GameObject, consentendo uno sviluppo rapido ma al contempo stabile e preciso.

Analisi degli scripts

1. StateMachine.cs

- **Obiettivo**: Gestisce le transizioni di stato del robot all'interno del sistema.
- **Funzionalità Chiave:**
    - SetState(State newState): Cambia lo stato corrente del robot, eseguendo l'uscita dallo stato attuale e l'ingresso nel nuovo stato.
    - GetCurrentState(): Restituisce lo stato corrente del robot.
    - Update(): Metodo richiamato ad ogni frame che verifica lo stato corrente e esegue le azioni di quel particolare stato.

2. NavigationState.cs

- **Obiettivo**: Gestisce il comportamento di navigazione del robot da un punto all'altro.
- **Funzionalità Chiave:**
    - EnterState(): Imposta le condizioni iniziali quando il robot entra in questo stato.
    - Execute State(): Contiene la logica per muovere il robot lungo un percorso predefinito e gestisce la rilevazione di ostacoli o nemici.

3. PlanningState.cs

- **Obiettivo**: Pianifica un percorso dal punto corrente del robot alla destinazione utilizzando l'algoritmo A*.
- **Funzionalità Chiave:**
    - EnterState(): Avvia la pianificazione del percorso.
    - ExecutePlanning(): Coroutine che implementa la logica dell'algoritmo A* per trovare il percorso ottimale.

4. ArrivalState.cs

- **Obiettivo**: Gestisce le azioni del robot all'arrivo alla destinazione.
- **Funzionalità Chiave:**
    - EnterState(): Viene eseguito quando il robot raggiunge la destinazione. Logga l'evento, indicando l'arrivo.

    - ExecuteState(): Esegue una serie di operazioni di pulizia una volta che il robot ha raggiunto la sua destinazione.
        - Riconfigurazione della griglia di navigazione tramite GridManager, che include la rigenerazione della griglia e la rilevazione di nuove celle bloccate. Questo assicura che la griglia rifletta l'ultimo stato dell'ambiente.ExitState(): Logga l'uscita dallo stato di arrivo, segnalando la conclusione delle operazioni di arrivo.
- Transizione: Il robot passa automaticamente allo stato di StandbyState tramite stateMachine.SetState(new StandbyState(stateMachine)) dopo aver completato le operazioni di arrivo.

5. StandbyState.cs

- **Obiettivo**: Mantiene il robot in uno stato di attesa, pronto per ricevere nuove istruzioni.
- **Funzionalità Chiave:**
    - EnterState(): Logga l'entrata nello stato di attesa, segnalando che il robot è in attesa di input dall'utente.
    - ExecuteState():
        - Ascolta l'input dell'utente per impostare una nuova destinazione. Se l'utente preme il tasto Q, il robot passa allo stato di PlanningState, indicando che è pronta a pianificare un nuovo percorso verso la destinazione specificata.
        - Utilizza GetDestinationFromMouseClick() per ottenere la posizione della destinazione basata sui clic del mouse dell'utente, convertendo le coordinate del clic in posizioni nel mondo di gioco.
    - ExitState(): Logga l'uscita dallo stato di standby.

6.DroneExplore.cs

- **Obiettivo:** Gestisce l'esplorazione dell'ambiente da parte del drone.
- **Funzionalità Chiave:**
    - Monitora e mappa l'ambiente circostante in tempo reale.
    - Identifica e segnala gli ostacoli e i nemici al sistema di controllo del robot.

7.DroneSoundController.cs

- **Obiettivo:** Controlla l'emissione dei suoni prodotti dal drone, come il rumore dei rotori.
- **Funzionalità Chiave:**
    - Regola il volume e il pitch dei suoni in base alla velocità e alle azioni del drone.
    - Migliora l'esperienza utente attraverso un feedback sonoro realistico che riflette il comportamento del drone.

8.RotorAnimation.cs

- **Obiettivo:** Gestisce le animazioni dei rotori del drone.
- **Funzionalità Chiave:**
    - Modifica la velocità di rotazione dei rotori in base ai comandi di movimento.
    - Assicura che le animazioni dei rotori siano sincronizzate con le azioni fisiche del drone.

9. KalmanFilter.cs

- **Obiettivo**: Implementa il filtro di Kalman per migliorare l'accuratezza delle stime di stato.
- **Funzionalità Chiave:**
    - UpdateState(): Aggiorna lo stato basandosi sulle misurazioni e sul modello di controllo.
    - Predict(): Prevede lo stato futuro basandosi sulle informazioni correnti.

10. RobotController.cs

- **Obiettivo**: Centralizza il controllo delle funzioni del robot, gestendo le operazioni di movimento e rilevazione.
- **Funzionalità Chiave:**
    - SetMoving(bool moving): Abilita o disabilita il movimento del robot.
    - RotateToTarget(Vector3 targetPosition) e MoveToTarget(Vector3 targetPosition): Gestiscono la rotazione e il movimento del robot verso un obiettivo specificato.
    - DetectDynamicEnemies(): Rileva la presenza di nemici dinamici entro un certo raggio.

11. GridManager.cs

- **Obiettivo**: Gestisce la creazione e la manipolazione della griglia di navigazione in cui il robot si muove.
- **Funzionalità Chiave:**

- GenerateGrid(): Crea la griglia basandosi su specifiche dimensioni e la rende interattiva mappando le posizioni mondiali delle celle.
- DetectBlockedCells(): Identifica e marca le celle bloccate in base alla presenza di ostacoli fisici come nemici o oggetti.
- MarkObjectsAsBlocked(GameObject[] objects, float radius): Marca le celle come non percorribili attorno agli oggetti specificati.

12. Cells.cs

- **Obiettivo**: Rappresenta una singola cella nella griglia di navigazione.
- **Funzionalità Chiave:**
  - GetWorldPosition(): Restituisce la posizione mondiale della cella.
  - IsWalkable(), SetWalkable(): Gestisce lo stato di percorribilità della cella.

13. AStar.cs

- **Obiettivo**: Implementa l'algoritmo A* per il calcolo del percorso più breve in un ambiente grigliato.
- **Funzionalità Chiave:**
  - FindPath(): Trova il percorso più breve tra due celle della griglia, considerando le celle bloccate.
  - GetNeighbors(Cell cell): Ottiene le celle vicine di una cella specifica, utilizzate per espandere il percorso nell'algoritmo A*.

14. MinimapFogOfWar.cs

- Obiettivo: Gestisce la logica della "fog of war" sulla minimappa, rivelando aree man mano che il robot le esplora.
- Funzionalità Chiave:
  - RevealArea(Vector3 playerPos): Rivela parti della minimappa in base alla posizione del robot.
  - UpdateDroneMarkerPosition(Vector3 playerPos): Aggiorna la posizione del marker del drone sulla minimappa.

15. Environment.cs

- **Obiettivo**: Configura e gestisce l'ambiente in cui il robot opera, incluse le aree con nemici.
- **Funzionalità Chiave:**
  - ResetEnemies(bool randomPosition): Reimposta le posizioni e lo stato dei nemici, eventualmente in posizioni casuali.

- GetEnemyFromCollider(Collider collider): Recupera e gestisce gli oggetti nemico associati a collider specifici.

16. Enemy.cs

- **Obiettivo:** Gestisce le caratteristiche e le azioni di un singolo nemico all'interno dell'ambiente di gioco.
- **Funzionalità Chiave:**
  - Found(): Evidenzia il nemico quando viene scoperto dal robot.
  - ResetEnemy(): Reimposta lo stato del nemico per una nuova sessione di gioco, rimuovendo eventuali evidenziazioni o modifiche di stato.

17. EnemyPatrol.cs

- **Obiettivo**: Gestisce il movimento pattuglia di entità nemiche, permettendo movimenti circolari o casuali dentro un'area definita.
- **Funzionalità Chiave:**
  - Update(): Aggiorna la posizione del nemico basandosi su un percorso predefinito o generato casualmente.
  - RotateTurret(): Gestisce la rotazione di eventuali torrette o armi del nemico, mirando in direzioni casuali.

18. WaypointsMovement.cs

- **Obiettivo**: Controlla il movimento di un oggetto (ad esempio un nemico) lungo una serie di waypoint predefiniti.
- **Funzionalità Chiave:**
  - Update(): Muove l'oggetto da un waypoint all'altro e gestisce pause tra un waypoint e l'altro.
  - AvoidObstacles(): Implementa una logica di evasione per gli ostacoli rilevati lungo il percorso**.**

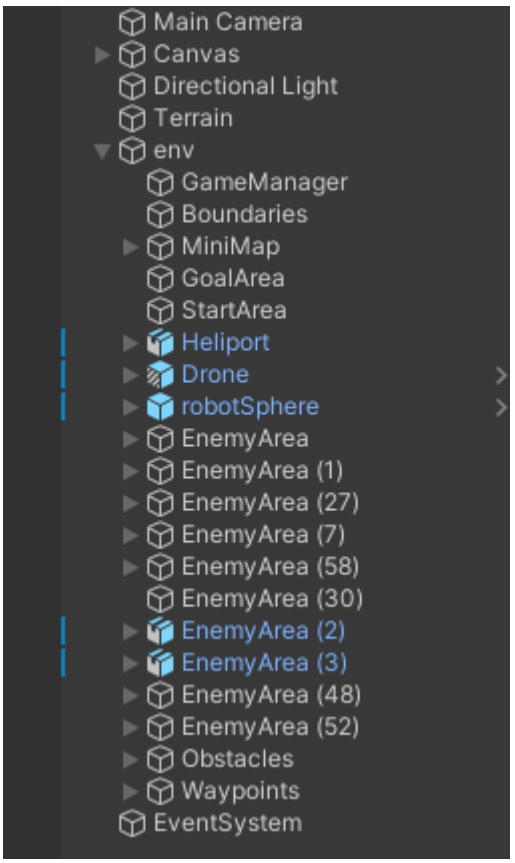
19. EnemyOutline.cs

- **Obiettivo**: Crea un contorno visivo attorno agli oggetti nemico per aumentarne la visibilità.
- **Funzionalità Chiave:**
  - CreateOutline(): Crea un duplicato dell'oggetto nemico con materiali specifici per renderlo visibile come contorno.

20. PathDrawer.cs

- **Obiettivo**: `PathDrawer.cs` è progettato per visualizzare graficamente i percorsi calcolati in un ambiente di gioco, usando Unity. Questo script è fondamentale per rendere visibili ai giocatori o agli sviluppatori i percorsi che gli agenti, come robot o personaggi, percorreranno, aumentando la chiarezza e l'interattività della navigazione nel gioco.
- **Funzionalità Chiave:**
    - DrawPath(List<Cell> path): Inizializza e inizia la visualizzazione del percorso. Interrompe eventuali animazioni precedenti se già in corso, configurando un nuovo `LineRenderer` se necessario e avviando la coroutine per animare il percorso disegnato.
    - SetupLineRenderer(): Configura il `LineRenderer` con proprietà come colore, larghezza della linea, e materiale, preparandolo a disegnare il percorso definito.
    - AnimatePathSequence(): Gestisce la sequenza completa di animazione del percorso, dalla visualizzazione alla dissolvenza, garantendo una transizione visiva fluida e professionale.

Scene



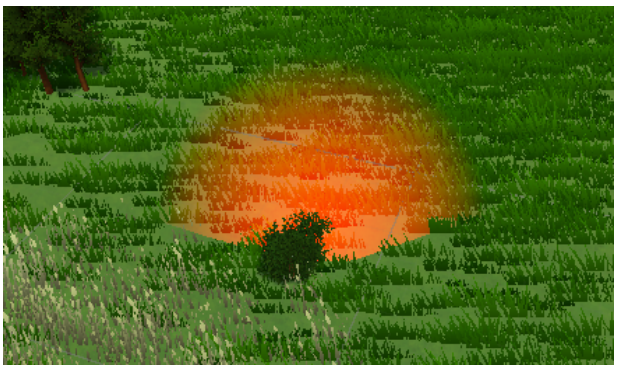
Il progetto prevede una main scene di cui fanno parte due macro objects.

Il terrain e l'environment.
Il **terrain** racchiude al suo interno tutti i gameobject tra cui l'environment.
L'environment comprende:
Minimap: Minimappa che mostra l'area di gioco vista dall'alto e rimpicciolita.

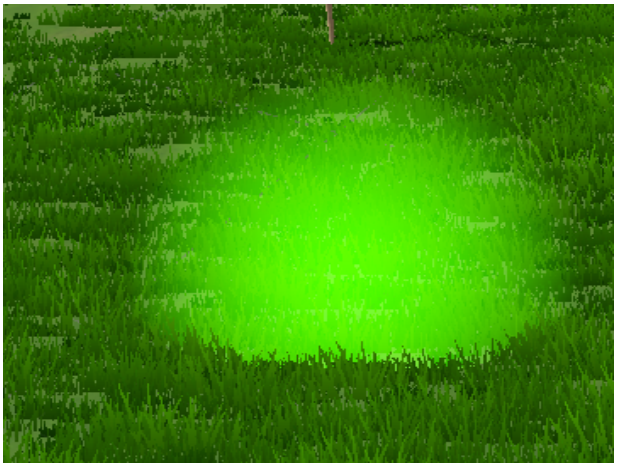
Esempio di minimappa oscurata dalla fog of war prima che il drone esploratore venga richiamato:

Esempio di minimappa dopo che il drone esplora ed evidenzia i nemici trovati.

Goal Area: l'area di arrivo del robot di terra



Start Area: l'area di partenza del robot di terra



Heliport: Stazione di partenza del drone esploratore

Drone: GameObject del Drone
RobotSphere: GameObject del robot di terra
Enemy Area in cui sono presenti i nemici.

Obstacles: GameObjects ostacoli quali rocce, alberi edifici.

DRONE

Nel progetto simulato, il drone funge da componente chiave, essenziale per la mappatura dell'ambiente, la ricognizione e il supporto al robot terrestre. Utilizzando una combinazione di tecnologie avanzate, il drone ottimizza le operazioni di ricognizione e facilita il superamento degli ostacoli, rendendo le missioni più sicure ed efficienti.

Design e Caratteristiche Tecniche

Il drone è progettato con un telaio robusto e leggero, equipaggiato con più rotori che gli conferiscono stabilità e manovrabilità eccezionali. La configurazione dei rotori, controllata dal modulo RotorAnimation, permette variazioni di velocità e inclinazione, adattandosi dinamicamente alle esigenze di volo e ai cambiamenti ambientali.

Capacità Sensoriali e Navigazione

Il modulo DroneExplore implementa l'intelligenza del drone, utilizzando sensori avanzati per una percezione dettagliata dell'ambiente:

- **Sistema Lidar**: Per la mappatura 3D precisa dell'ambiente, fondamentale nella navigazione e identificazione di ostacoli.
- **Sistema di Navigazione Intelligente**: Utilizza algoritmi ML per apprendere e ottimizzare percorsi di volo in vari scenari.

Funzioni Principali

- **Ricognizione Avanzata**: Identifica ostacoli e nemici in modo preciso attraverso l'utilizzo del filtro di Kalman, fornendo dati critici al robot terrestre per la pianificazione del percorso.
- **Mappatura Dettagliata**: Crea mappe dettagliate dell'ambiente che sono essenziali per la navigazione e la strategia complessiva delle missioni.
- **Supporto Operativo**: Interviene in scenari complessi dove è necessaria una visualizzazione aerea o un intervento rapido.

Apprendimento e Adattamento

Un aspetto cruciale del funzionamento del drone nel progetto è la sua capacità di apprendere e adattarsi a vari scenari operativi attraverso un processo di apprendimento automatizzato. Utilizzando il framework di Unity ML-Agents, il drone è addestrato in un ambiente simulato che riproduce fedelmente le condizioni operative reali ma con configurazioni variabili degli ostacoli e delle minacce.

Simulazione per l'Apprendimento

Durante la fase di training, il drone è stato immerso ripetutamente in scenari dove la disposizione dei nemici e la configurazione degli edifici variavano in modo casuale. Questo approccio ha permesso al drone di sperimentare un'ampia gamma di situazioni potenziali, dalle più semplici alle più complesse, affinando le sue strategie di navigazione e decisione basate sui dati sensoriali raccolti in tempo reale.

Tecniche di Training

Il processo di training utilizza tecniche di apprendimento per rinforzo, dove il drone riceve feedback immediato (reward) basato sulle sue azioni. Ad esempio, evitare con successo un ostacolo o identificare correttamente un nemico porta a un reward positivo, mentre l'impatto con un ostacolo o il mancato rilevamento di una minaccia comporta una penalità. Questo sistema di reward e punishment aiuta il drone a ottimizzare le proprie scelte in modo autonomo.

Ottimizzazione Continua

Grazie a questa continua esposizione a variabili ambientali mutevoli, il drone migliora la sua capacità di prevedere e reagire a cambiamenti improvvisi nell'ambiente. L'algoritmo di apprendimento si affina iterazione dopo iterazione, migliorando non solo l'efficienza del percorso di volo ma anche la precisione con cui vengono eseguite le operazioni di ricognizione e mappatura.

Robot di Terra

Il robot terrestre nel nostro progetto di simulazione svolge un ruolo cruciale nella navigazione dell'ambiente di simulazione, evitando ostacoli e interagendo con vari elementi dinamici. Dotato di avanzati sistemi sensoriali e una robusta capacità di calcolo per il pathfinding, il robot rappresenta la componente chiave per le operazioni a terra.

Capacità e Funzionalità

Il robot è equipaggiato con un sofisticato sistema di controllo che gestisce movimenti e interazioni ambientali. Utilizzando il modulo RobotController, il robot può navigare autonomamente verso la destinazione impostata, rilevare nemici e ostacoli dinamici, e adattare il suo percorso in risposta a cambiamenti dell'ambiente attraverso l'algoritmo di navigazione A*.

Rilevamento e Navigazione

La capacità di rilevare **dinamicamente** i nemici è fondamentale per la sicurezza del robot. Utilizzando un raggio di rilevamento, il robot identifica nemici nell'immediata vicinanza, aggiorna la griglia con l'ausilio dell'algoritmo A* e **rigenera** il percorso corretto per evitare collisioni e intrappolamenti. Questo sistema non solo aumenta la sopravvivenza del robot ma anche **l'efficacia** della missione complessiva.

Interazione con il Drone

Il robot collabora strettamente con il drone, ricevendo dati essenziali attraverso l'aggiornamento della griglia per la navigazione e l'aggiornamento del percorso. Questa sinergia tra le due unità aumenta notevolmente l'efficacia delle operazioni, permettendo al robot di prendere decisioni informate basate su una ricca percezione ambientale fornita dal drone.

Filtro di Kalman

Nel nostro progetto Unity, il Filtro di Kalman è stato utilizzato per affinare la precisione della localizzazione dei nemici, migliorando così la capacità del sistema di simulazione di rispondere in modo accurato e tempestivo alle minacce. Questo algoritmo è cruciale per ottimizzare le stime dello stato dei nemici basate su misurazioni imprecise.

Inizializzazione del Filtro

Il filtro inizia con la definizione delle variabili di stato, che in questo caso sono le coordinate spaziali x, y e z del nemico. Le matrici di covarianza dello stato (P), il rumore di processo (Q) e il rumore della misura (R) vengono inizializzate per configurare la sensibilità e la reattività del filtro alle dinamiche del gioco. La matrice identità (I) serve come base per le operazioni matriciali successive.

Configurazione delle Matrici

- **Covarianza dello stato (P):** Inizialmente impostata come una matrice identità, questa matrice rappresenta l'incertezza iniziale sulla stima dello stato.
- **Rumore di processo (Q):** Configurata per riflettere l'incertezza nel modello di movimento dei nemici, con valori distinti per ogni dimensione dello spazio, permettendo al filtro di adattarsi alle diverse volatilità nei movimenti.
- **Rumore della misura (R):** Stabilito in base all'attendibilità delle misurazioni sensoriali, influenzando quanto il filtro dovrebbe "fidarsi" delle nuove informazioni ricevute.

Fase di Predizione

Durante la predizione, il filtro proietta in avanti la stima corrente dello stato basandosi esclusivamente sul modello interno, senza considerare nuove misurazioni. Questo passaggio aggiorna la covarianza dello stato, aumentandola di un valore proporzionale al rumore di processo per riflettere l'aumento dell'incertezza nel tempo.

Fase di Aggiornamento

Quando vengono acquisite nuove misurazioni, il filtro le utilizza per aggiornare la stima dello stato:

- **Calcolo del Residuo:** Determina la differenza tra la misurazione effettiva e la stima predetta.
- **Aggiornamento dello Stato e della Covarianza:** Utilizza il guadagno di Kalman, calcolato dalla relazione tra la covarianza dello stato e il rumore della misura, per ponderare quanto aggiornare la stima dello stato in base alla nuova misurazione. La covarianza dello stato è quindi aggiornata per riflettere la nuova certezza dopo l'integrazione della misurazione.

L'implementazione del Filtro di Kalman nel progetto ha quindi significativamente migliorato la **precisione** della **localizzazione** dei nemici, rendendo il sistema di simulazione più **affidabile** e efficace. Questa tecnica ha permesso di mantenere un equilibrio tra reattività e stabilità nelle stime, ottimizzando l'interazione tra il drone e i nemici in scenari dinamici e imprevedibili.

GridManager: Strutturazione dell'Ambiente

Il GridManager è il cuore della **rappresentazione** dell'ambiente nel nostro progetto. Gestisce una griglia di celle che copre l'area operativa, con ogni cella che rappresenta una porzione dello spazio fisico. Ogni cella, definita dall'oggetto Cell, può essere contrassegnata come percorribile o bloccata a seconda della presenza di ostacoli o nemici.

Funzioni Principali del GridManager:

- **Generazione della Griglia**: Crea una griglia basata su dimensioni predefinite, inizializzando ogni cella come percorribile e poi aggiornandola in base agli ostacoli rilevati.
- **Rilevamento delle Celle Bloccate**: Identifica le celle che contengono ostacoli fisici o nemici, marcandole come non percorribili per evitare che il robot tenti di attraversarle.
- **Supporto alla Navigazione**: Fornisce dati essenziali al sistema di pathfinding A* per la determinazione del percorso ottimale.

Algoritmo A: Calcolo del Percorso*

L'algoritmo A*, implementato nel modulo AStar, è responsabile del calcolo del **percorso più breve** e sicuro dalla posizione iniziale del robot a quella finale. Utilizza una combinazione di **costi di movimento** (G-cost) e euristiche (H-cost) per determinare il percorso più efficiente attraverso la griglia.

Processo di Pathfinding con A:*

- **Inizializzazione**: Partendo dalla cella iniziale, l'algoritmo aggiunge celle alla lista aperta e le sposta nella lista chiusa man mano che vengono esplorate.

- **Calcolo dei Costi**: Ogni cella ha un costo di movimento associato e un costo euristico che stima la distanza dalla cella di destinazione.
- **Ottimizzazione del Percorso**: L'algoritmo cerca il percorso con il costo totale minimo, aggiornando i percorsi potenziali con nuove scoperte fino a raggiungere la destinazione.

Integrazione di GridManager e A nel Controllo del Robot

La stretta integrazione tra il GridManager e l'algoritmo A* permette al robot di **adattarsi** dinamicamente a cambiamenti dell'ambiente, come la comparsa di nuovi ostacoli o nemici. Questa capacità di adattamento è fondamentale per mantenere alta **l'efficienza** e la **sicurezza** delle operazioni del robot.

Possibili applicazioni al mondo reale

Un'applicazione reale di questo tipo di tecnologia potrebbe trovarsi nel campo della logistica e delle operazioni di soccorso in scenari di emergenza. Ad esempio, il sistema di collaborazione tra drone e robot terrestre potrebbe essere utilizzato per la navigazione e il soccorso in aree colpite da disastri naturali, come terremoti o inondazioni, dove la mappa degli ostacoli e delle vie praticabili può cambiare rapidamente e dove l'accesso diretto può essere pericoloso o impossibile per gli umani.

Scenario di Emergenza

In queste situazioni, i droni potrebbero essere inviati per sorvolare l'area colpita e raccogliere dati in tempo reale sulla distribuzione di detriti, ostacoli, zone allagate o incendi attivi. Queste informazioni verrebbero poi utilizzate per aggiornare le mappe utilizzate dai robot terrestri. I robot, a loro volta, potrebbero navigare in modo autonomo attraverso le vie più sicure per raggiungere persone intrappolate o per consegnare aiuti, come cibo, acqua o medicinali.

Applicazioni Logistiche

In ambito logistico, un sistema simile potrebbe essere impiegato in magazzini di grandi dimensioni o in aree portuali per il trasporto di merci. I droni potrebbero monitorare e gestire il flusso di veicoli e merci, identificando i percorsi ottimali e riducendo il tempo di trasporto all'interno di grandi infrastrutture logistiche, mentre i robot terrestri eseguirebbero le operazioni di movimentazione in base alle indicazioni ricevute, ottimizzando l'efficienza operativa e minimizzando il rischio di incidenti o intasamenti.

Questi esempi dimostrano come la collaborazione tra droni e robot terrestri possa essere sfruttata per migliorare la sicurezza, l'efficienza e la rapidità di risposta in

diversi ambienti e contesti operativi, sfruttando le capacità uniche di ciascuna tecnologia per superare le limitazioni dell'altra

Conclusione

Lo sviluppo del progetto è stato terminato nei tempi stimati e con il raggiungimento di quasi tutti gli **obiettivi** prefissati.

Durante il progetto, il team ha affrontato numerose **sfide**, dalle questioni tecniche legate all'algoritmo di navigazione alla gestione degli imprevisti ambientali che influenzavano i sensori del drone. In ogni occasione, i membri del gruppo hanno **collaborato** strettamente, utilizzando sessioni di brainstorming e discussioni approfondite per ideare **soluzioni innovative** che hanno permesso di superare gli ostacoli incontrati.

Questa esperienza ha non solo rafforzato le **competenze tecniche** individuali, ma ha anche arricchito la capacità di ogni membro di lavorare **efficacemente** in un ambiente di team.