Tecnológico de Monterrey

Evidence 2. Progress and Presentation of the Challenge

Review 3

Cesar Moran Macias | A01645209

Ana Paola Jiménez Sedano | A0164453

Ariana Guadalupe Rosales Villalobos | A01644773

Demmí Elizabeth Zepeda Rubio | A01709619

Luisa Merlo García | A01067715

Modeling of Multi-Agent Systems with Computer Graphics

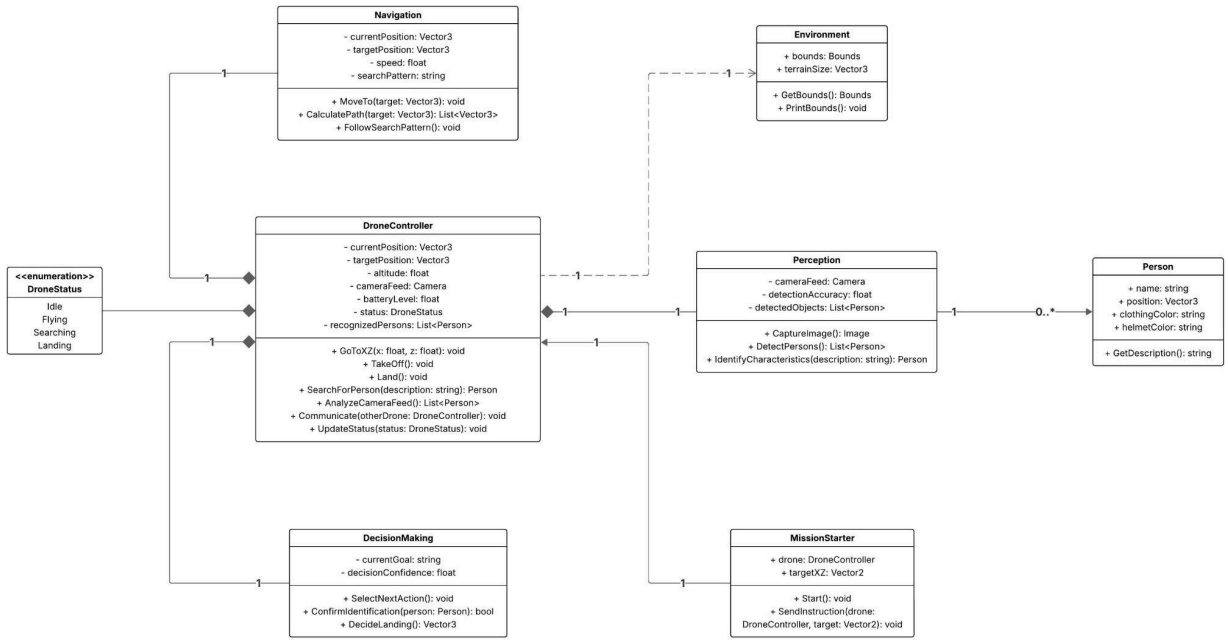
Professor Iván Axel Dounce Nava

September 9th, 2025

Final UML Agent Class Diagrams & Interaction Protocols

Class Diagram

The following diagram is a class diagram to help explain the behavior of the simulation with the navigation agent. This is the link for better visualization.



Interaction Protocols

Computational Vision Agent

The following is an interaction protocol for the computational vision agent. This is the link for better visualization.

SubAgent/ computationalVision: MAVAgent

Attributes: DataType
knowledge_base : OntologyPersonDescription
person_description: OntologyPersonDescription
current_location : (x, y, z)
current_altitude : alt
probability_match : percentage
visual_data: ImageData

Goals
permanentGoals: wff={ "find person of interest"}
actualGoals: wff ={ "Extract characteristics of current object observed"
"Compare observed characteristics with the ones in knowledgeBase"}

Actions

1. verifyTargetDescription(visual_data: ImageData): current_location and probability_match <<pro-active>>
- Pre: The agent has visual data from the environment, possibly containing people.
- Post: Observed traits are processed to be compared with the person of interest traits.
- Effects: A probability match and the position in which it is, is returned
- Invariants: The matching process must follow the odontology criteria of OntologyPersonDescription.

2. targetFoundInform(current_location: (x,y,z),probability_match) — <<pro-active>>
- Pre: The observed characteristics matched with the person of interest at mostly 90%.
- Post: An external agent recieves a message
- Effects: The agent sends an "inform" message (declaring that the target may be in that position) to the master
- Invariants: The message should use words from the ontologies and in string format.

3. notUnderstood(person_description: OntologyPersonDescription) - <<re-active>>
- Pre: The agent has received a message it cannot interpret.
- Post: The original sender is notified of the misunderstanding.
- Effects: The agent signals a comprehension failure.
- Invariants: The not_understood message must clearly state what it didn't understood.

4. reportObstacles(obstacle_details: ObstacleInfo) -<<re-active>>
- Pre: The agent has detected an obstacle via its camera and has relevant details (area coordinates and type).
- Post: An inform communicative act has been sent to the navigation agent with the obstacle details.
- Effects: The navigation knowledge base is updated with the obstacle information, allowing it to adjust plans or behaviors.
- Invariants: The inform message must clearly and precisely describe the obstacle using a shared ontology.

Methods

1. interpretMission(person_description) → void
- Pre: Has recieved PersonOfInterest_description
- Post: Description is saved on knowledgeBase
- Effects: The MAV's knowledge base is updated with mission details.
- Invariants: The message is only in terms of the ontologies

2. scanAreaForPeople() → visual_data
- Pre: The MAV is in the designated search area.
- Post: The agent generates its percepts.
- Effects: The agent collects visual perception data about the current object.
- Invariants: The camera is configurated correctly (angle, altitude and color detection)

3. updateInternalState() → void
- Pre: The agent has received new perceptual inputs (traits observed) from the environment.
- Post: The agent's internal state incorporates the new perceptions.
- Effects: Internal state model is updated
- Invariants: The perceptual inputs need to be under the ontologyPersonDescription

Capabilities:
- Visual perception of the environment (substraction of traits from image)
- Receive the person of interest description from the SWARM master and save it in its knwoledge base in comprehensible terms
- Comparision of observed traits and desired traits
- Find the person of interest and report it to SWARM master
- Report obstacles to agent navigation

Constraints
1.- Society-name: PersonIdentificationProtocol
[constraint]: [confidence_score > 0.90]
2.-society-name: ObstacleReportingProtocol
[constraint]: [obstacle_type = 'obstructs_trajectory']
3.-society-name: SearchPersonOfInterestDescription
[constraint]: [gender = 'Feminine' AND clothing_top = 'Black blouse' AND clothing_bottom = 'White skirt']

<<Agent_computationalVision>> Head Automata
States
- **Idle**, input: wait for mission
- **MissionLoaded**, input: interpretMission())
- **Scanning**, input: scanAreaForPeople() → visual_data
- **Matching,** input: verifyTargetDescription(visual_data) → (current_location, probability_match)
- **CandidateFound**, input: inform("candidate at (x,y,z), probabilityMatch=...")
- **Confirming**, input: wait for master confirmation of target double check
- **NotUnderstood,** input: not_understood(received_message)

Transitions
Perceive → Transition / Entry action
- Idle → MissionLoaded: inform(person_description) / interpretMission()
- MissionLoaded → Scanning/ scanAreaForPeople()
- Scanning → Matching: compare(knowledgeBase and internalState)/ updateInternalState(visual_data)
- Matching → CandidateFound: [probability_match ⩾ 0.90] / inform(targetFoundInfo)
- Matching → Scanning: [probability_match < 0.90] / continue_scan
- CandidateFound → Confirming: inform(sendMessageToSWARM) / await master response
- Confirming → LandingRequestIssued: inform(confirmation_ok) / waitLandingInChargeOfNavigation
- Any → NotUnderstood: Message(not parseable) / not_understood(message)

Navigation Agent

The following is an interaction protocol for the navigation agent used. This is the link for better visualization.

<<Navigation_Agent>

Attributes: DataType
knowledgeBase : OntologyNavigation
current_location : (x, y, z)
target_location:(x,y,z)
Searchpattern:
current_altitude : alt
current_velocity: m/s
current_position: Coordinates
Search_State: Active, stopped, paused
flight_state: (Onroute, search, align, land)

Permanent goals: wff = {"Navigate until you find the target person", "Avoid obstacles reactively", "Update status and critical events", "Register current state", "Communicate actively with the system"
}

actualGoals: wff ={ "Sets the course when the target is found", "Ascend", "Search with a pattern strategy", "Request for alignation within the target", "Confirms the absence of obstacles", "Adjustments for descent", "Validates the safe distance between the drone and the target
"
}

Actions

StartSearch <Re-active> (area: Polygon, pattern: SearchPattern)

Pre: The target area is near the drone's arrival.
Post: A grid, spiral, or zigzag coverage plan is instantiated across the region.
Effects: The agent begins executing patterns and enters the "Search" stage.

Setcourse (gps_target) <Re-active>
Pre: The mission's destination has been announced by the swarm master.
Post: A path is created in the direction of the destination.
Effects: The agent updates the current state after switching to the "On Route" state.

Executes_PatternSearch(waypoint: Vector3) <Proactive>

Pre: The next waypoint is defined and a search pattern is in effect.
Post: The MAV collects visual data as it approaches the waypoint.
Effects: Perception-related new poses and frames are created, and the coverage percentage is modified.
Invariants: Vertical and lateral velocity cannot be higher than permitted.

reportObstacles(obstacle_details: ObstacleInfo)

Pre: Using sensors and a camera, the agent has identified an obstacle and has the necessary information (area coordinates, type, extent).
Post: The impediment specifics are given along with an informative communicative act.
Effects: Plans and actions can be modified thanks to the enhanced navigation knowledge base.

VerifyTarget_Description <pro-active> (visual_data: ImageData): probability_match and current_location

Pre: The agent has visual information from the surroundings, which may include human beings.
Post: The person-of-interest qualities are compared with the observed traits after processing.
Effects: The candidate's estimated position and a probability match are returned.
Invariants: The matching procedure needs to adhere to the OntologyPersonDescription criteria.

TargetFoundInform(current_location: (x,y,z), probability_match) <pro-active>

Pre: There was a chance of ⩾ 0.90 that the observed features matched the person of interest.
Post: A message is sent to an external agent.
Effects: The agent notifies the swar, master that the target might be in that location by sending a "inform" message.

startLandingSequence <re-active> (approved: LandingZone)

Pre: The landing has been confirmed by the Swarm master.
Post: We follow the protocol of landing Align → Validate → Descent.
Effects: The agent starts precise landing actions after exiting search mode.
Invariants: Checks and sequence order must be adhered to The Landing Protocol.

Replan_Landing(offset: Vector2) <re-active>

Pre: The master prevented or rejected a landing attempt.
Post: Using the given offset or limitations, a new landing zone is calculated. The result is a new *LandingPoint*.
Invariants: Unsafe zones that have already been rejected cannot be reused during the replan.

Methods

Compute_Route_Target(gps_target) → WaypointList
Pre: A destination for the mission has been determined.
Post: From the present stance to the target area, an efficient waypoint sequence is generated.
Effects: Speed and altitude profiles are used to build the navigation plan.
Invariants: Routes adhere to no-fly zones and geofences.

Generate_SearchPattern → WaypointList (area: Polygon, pattern: SearchPattern)

Pre: Near the search area, the MAV has arrived.
Post: Area-wide waypoints (grid, spiral, and zigzag) are created.
Effect: Using the ground sampling distance and field of view as parameters, a coverage plan is instantiated.

EvaluateHypothesis → MatchScore

Pre: For the current frame, a detection hypothesis is present.
Post: The person description is compared to a similarity score.
Effects: A single probability or score is created by combining matching features.
Invariants: weighting and thresholds.

ValidateLandingZone → SafetyCheck

Pre: There is a landing zone for candidates.
Post: For the following reasons (slope, obstacle density, and person distance), SafetyCheck returns OK/Reject.
Effects: The landing zone is either approved or denied for descent.

computeDescentProfile → DescentPlan (LandingZone)

Pre: SafetyCheck=OK is present in the landing zone.
Post: Hold points and a vertical speed profile are generated.
Effects: The restrictions are defined in the descent plan.

Execute_descent -> DescentStatus (Landingprotocol)

Pre: Alignment requirements are satisfied and safety is acceptable.
Post: Altitude is decreased as planned.
Effects: The MAV periodically performs safety checks as it approaches to the landing zone.

Handle_Abort → (reason: AbortReason)

Pre: There is a violation of a critical feature.
Post: Failure is reported once the MAV reaches a safe height and holds.
Effects: A fresh decision is awaited and current activities are posponed.

Capabilities :

- Visual perception of the surroundings.
- Get the SWARM master's description of a Person-of-Interest and enter it into the knowledge base.
- Determine a match score by comparing intended and observed qualities of the target.
- Notify the SWARM master of any candidate sightings and locate the Person-of-Interest.
- Identify, categorize, and report obtacles; then, adjust the navigation map appropriately.
- Create and follow routes to a GPS target; use search patterns.
- Perform a controlled descent after aligning over the authorized landing area.

Constraints

- society-name: ReplanProtocol [constraint]: [reused_blocked_landingzone = false]
- LandingProtocol [constraint]: [distance_to_person_m >= 1.0 AND distance_to_person_m <= 2.0]
- society-name: ObstacleReportingProtocol [constraint]: [obstacle_type = obstructs_trajectory]

<Agent_Navigation> Head Automaton

States

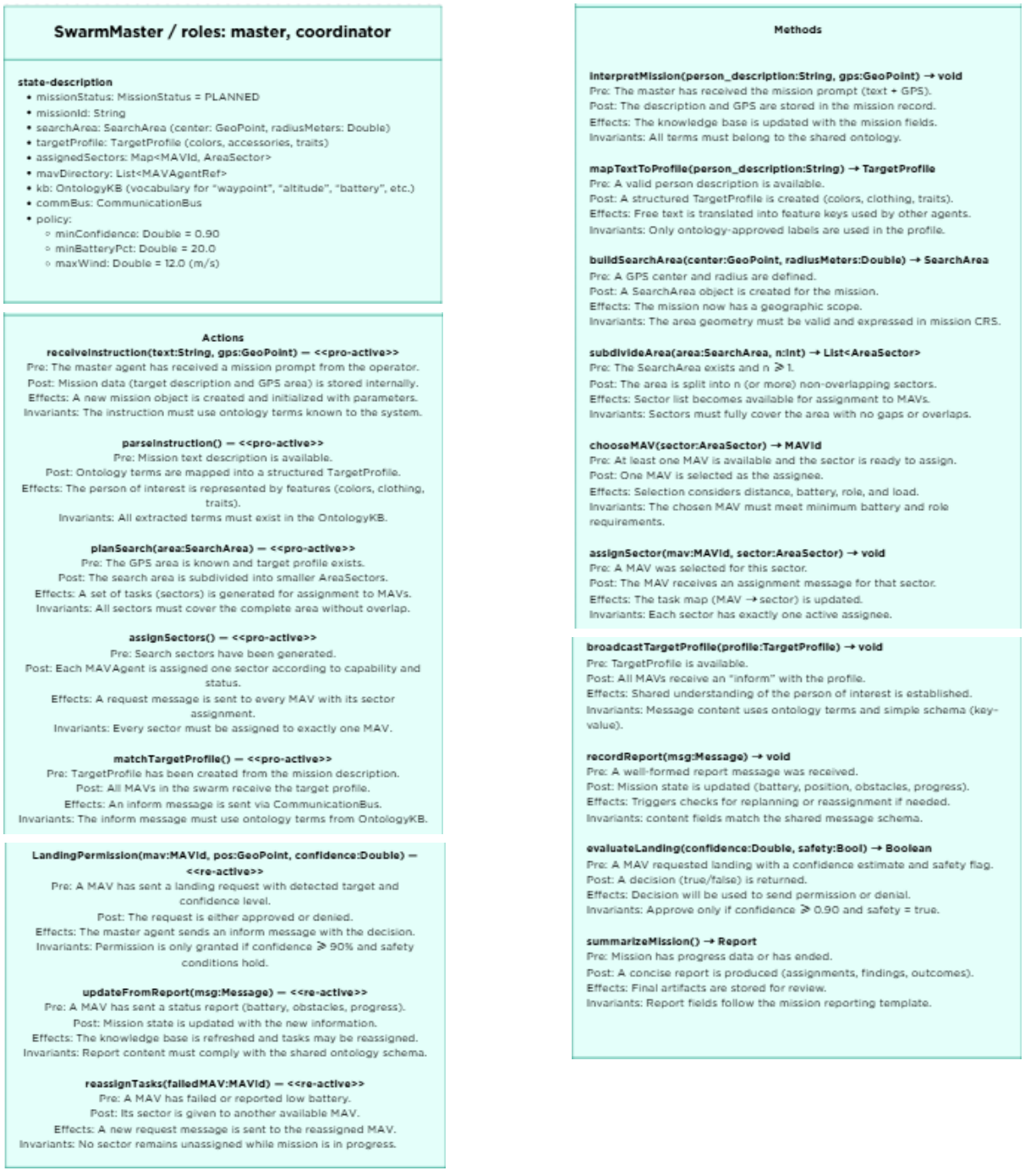
- MissionLoaded, input: interpretMission(person_description)
- Scanning, input: scanAreaForPeople() → visual_data
- Matching, input: verifyTargetDescription(visual_data) → (current_location, probability_match)
- CandidateFound, input: targetFoundInform(current_location, probability_match)
- Confirming, input: wait for master confirmation
- LandingRequestIssued, input: startLandingSequence(approved_lz)

Transitions

- Idle → MissionLoaded: receive(mission(person_description)) / interpretMission()
- Matching → TargetPersonFound: [probability_match ⩾ 0.90] / targetFoundInform(current_location, confidence)
- Scanning → Matching: [new_frame / updateState(visual_data)
- CandidateFound→ Confirming: inform(sendMessageToSWARM(candidate))

SWARM Master Agent

The following is an interaction protocol for the SWARM Master Agent. This is the link for better visualization.



Capabilities

- Receive and interpret the mission (description of the person + GPS location).
- Decompose the mission into subgoals and assign them to each MAV agent.
- Coordinate communication between agents using shared ontologies.
- Maintain an internal map of agent roles, areas, and progress.
- Evaluate confirmation messages from agents and validate landing decisions.
- Reassign tasks if communication fails or conflicts occur.

Service Description

- Acts as a centralized coordinator in the swarm.
- Ensures consistency across knowledge bases.
- Provides each MAV with its specific sub-area to search.
- Broadcasts the landing permission when target is confirmed.
- Collects reports of obstacles, misunderstandings, and detected targets.
- Validates messages based on ontological structure and expected protocols.

Supported Protocols

- **MissionDistributionProtocol:**
Used to send the mission to each MAV agent (GPS + Person Description).

- **TargetConfirmationProtocol**
Used by MAVs to inform the master about potential matches.
-
- **ObstacleReportingProtocol**
- Used to notify the master about obstacles that interfere with flight.

- **TaskAssignmentProtocol**
- Used to assign or reassign search zones.

- **LandingPermissionProtocol**
- Used when a MAV requests authorization to land near the identified target.

Constraints

1. Society-name: MissionDistributionProtocol
2. [constraint]: person_description != null AND gps_location != null
3. Society-name: TargetConfirmationProtocol
4. [constraint]: confidence_score >= 0.90 AND message_format == "ontology_compliant"
5. Society-name: ObstacleReportingProtocol
6. [constraint]: obstacle_type IN ["building", "tree", "drone", "restricted_zone"]
7. Society-name: LandingPermissionProtocol
8. [constraint]: distance_to_target <= 2m AND risk_level == "low"
9. Society-name: TaskAssignmentProtocol
10. [constraint]: agent_status == "idle" OR search_area == "unassigned"

- **<<Agent_computationalVision>> Head Automata**

States

- **Idle**, input: wait for mission
- **MissionLoaded**, input: interpretMission())
- **Scanning**, input: scanAreaForPeople() → visual_data
- **Matching,** input: verifyTargetDescription(visual_data) → (current_location, probability_match)
- **CandidateFound**, input: inform("candidate at (x,y,z), probabilityMatch=...")
- **Confirming**, input: wait for master confirmation of target double check
- **NotUnderstood,** input: not_understood(received_message)

Transitions

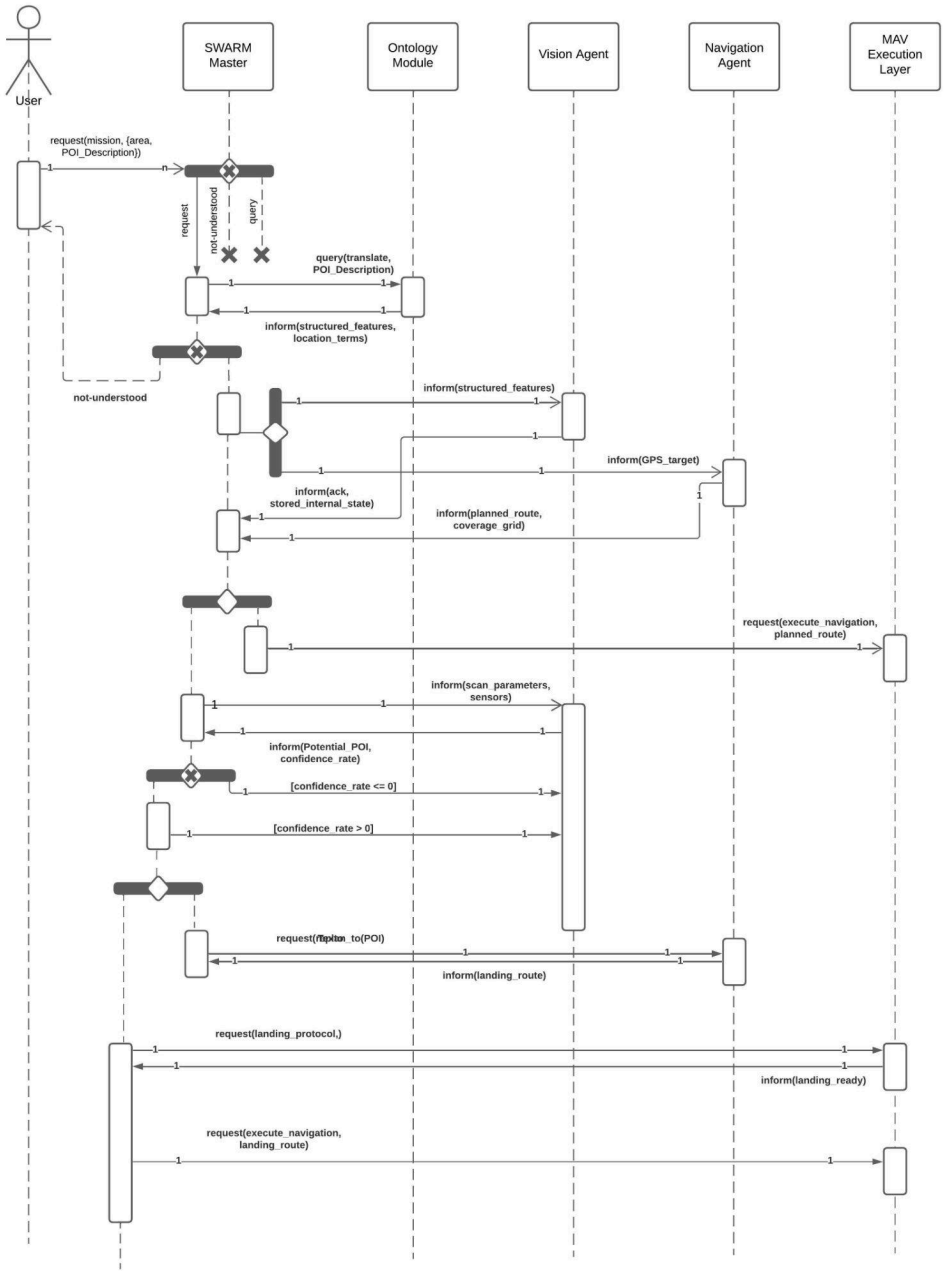
Perceive → Transition / Entry action

- Idle → MissionLoaded: inform(person_description) / interpretMission()
- MissionLoaded → Scanning/ scanAreaForPeople()
- Scanning → Matching: compare(knowledgeBase and internalState)/ updateInternalState(visual_data)
- Matching → CandidateFound: [probability_match ≥ 0.90] / inform(targetFoundInfo)
- Matching → Scanning: [probability_match < 0.90] / continue_scan
- CandidateFound → Confirming: inform(sendMessageToSWARM) / await master response
- Confirming → LandingRequestIssued: inform(confirmation_ok) / waitLandingInChargeOfNavigation
- Any → NotUnderstood: Message(not parseable) / not_understood(message)

Other UML Diagrams

Sequence Diagram

The following is a sequence diagram representing the simulation. This is the link for better visualization.



Agents Implementation

In this stage of the development, we have focused on implementing the core logic of the navigation agent, which is defined in the DroneController.cs script. Its responsibility is moving the drone toward a required coordinate in the environment, ensuring a safe operation by avoiding obstacles in its path. If the target position is identified as unsafe, the agent will instead land at a safe distance of two meters from that point, which guarantees the completion of the task without collisions.

The navigation logic is built on Unity's NavMesh system, allowing the calculation of routes from the drone's current position to the objective. It is the GoToXZ() function that is in charge of the route planification, projecting both the starting and destination point onto the navigation mesh, and then generating a sequence of waypoints. The waypoints are elevated to the defined cruising altitude to simulate flight, and during each update cycle, the method of FlyAlongPath() guides the drone step by step. Once it reaches the end, the landing logic is triggered through the TryLandNearMissionXZ() method. If the position is blocked, it will start a radial search around the zone with FindSafeLanding(). Said function samples several candidate points around the target and validates them through IsClearForLanding(); if the location is found safe, the drone will execute a controlled descent via LandCoroutine routine, otherwise, the agent maintains a searching for landing state.

Looking ahead, the next step in the development of this agent is to integrate its navigation capabilities with the vision agent. Once combined, the drone will not only fly to its assigned coordinate but also execute circular search patterns around the area to detect the person of interest. If a potential match is detected, the drone will communicate with the Swarm Master agent, which will authorize or deny a safe landing request based on the mission context.

Video of the simulation working

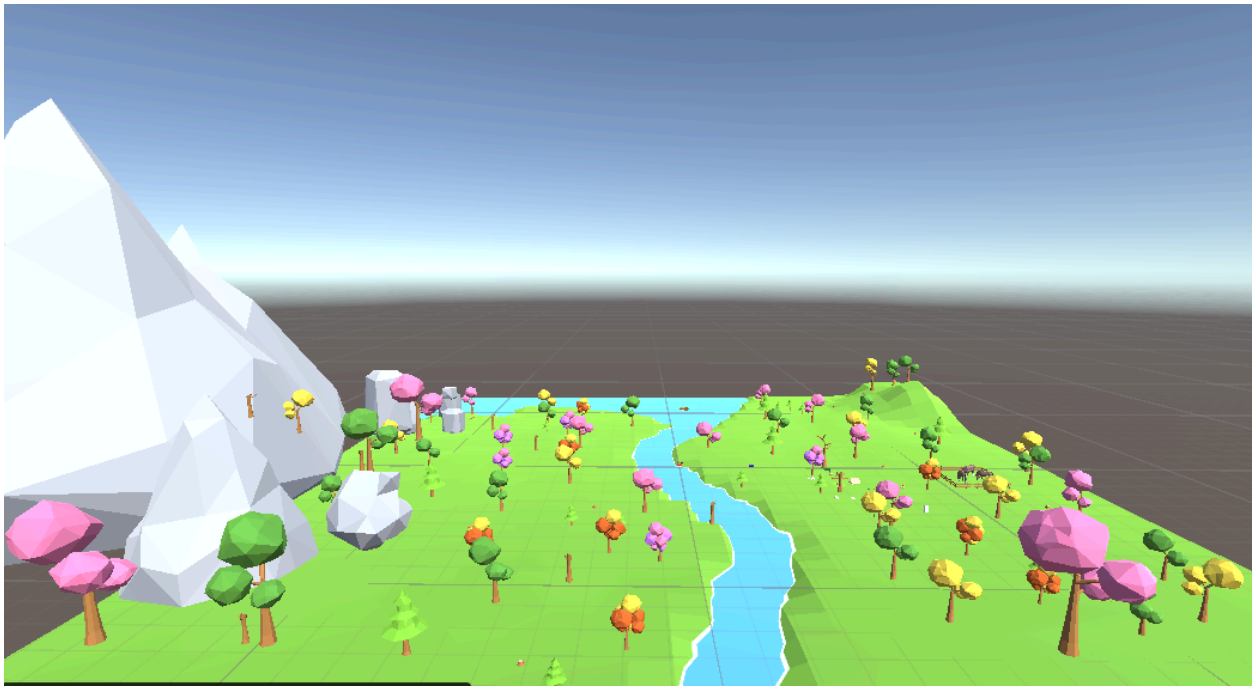
https://tecmx-my.sharepoint.com/:v:/g/personal/a01644532_tec_mx/EWwYthok_FZAioyUdFZmT7EBzs50WYNTaF_061BHA4_H6A?nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcml2ZUZvckJ1c2luZXNzIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXciLCJyZWZlcnJhbFZpZXciOiJNeUZpbGVzTGlua0NvcHkifX0&e=3ygo2Q

Graphical Implementation

The following are screenshots of the current development made in the graphical implementation of our solution for the challenge in Unity, with many objects and elements modeled in Blender.

This map represents the search and rescue challenge scenario in Unity, the environment is designed as a mission area where autonomous drones (MAVs) must operate, and the terrain is divided into several zones that the agents need to explore, identify, and navigate.

Key elements of the map include open areas for flight, pathways that guide the search patterns, and marked locations where the person of interest could be found, the map simulates real-world conditions, helping to test navigation, perception, and coordination among multiple agents in a controlled virtual space.



Work Plan

Completed Activities

Task	Responsible	Due Date	Given Effort
Finalize corrections of UML diagrams and interaction protocols.	Ariana & Luisa	28/08/2025	2 hours
Advancement on the code for the implementation of the visual agent	César	29/08/2025	3 hours
Advancement on the code for the implementation of the navigation agent	Ana	29/08/2025	3 hours
Advancement on the code for implementing the graphical solution	Demmí	29/08/2025	3 hours

Pending Activities

Task	Responsible	Due Date	Estimated Effort
Implementation of computational vision in code	César & Ariana	10/09/2025	2-3 hours
Connect the interface with the backend	Cesar & Luisa	10/09/2025	2 hours
Make final presentation of the project in Canva	Demmí	11/09/2025	1 hour
Polish and revise the final PDF	Demmí	11/09/2025	1 hour
Polish final details of the scripts	Luisa	11/09/2025	1-2 hours
Test the project before presentation	Ariana	11/09/2025	1 hour
Make a final revision of the	Ana	11/09/2025	1-2 hours

scripts and presentation			

Acquired Learning

Throughout the development of this phase, our team strengthened our collaborative and technical skills in the following fields.

- MAS Modeling: we deepened our understanding of the interactions between the different agents, with which we defined the diagrams to represent the systems behavior.
- Unity Implementation: we gained practical experience in integrating agents into Unity, as we set up the environment with assets, and connected the scripts to graphical elements.
- Agents: we learned to implement the navigation behavior, ensuring the agent can respond to specific coordinates and tasks while maintaining coherence to the simulation.
- Team Coordination: by dividing tasks, we improved our workflow and time management.

With this, we managed to develop a proper solution to the challenge given.