Tecnológico de Monterrey

Evidence 2

Victor Jaziel Coronado Flores A01644090

Diego Iván Morales Gallardo A01643382

Luis Daniel García Espinosa A01643058

Jorge Antonio Arizpe Cantú A01637441

Michael Rice Radilla A00837460

29 nov 2024

Modeling of Multi-Agent Systems with Computer Graphics

Prof. Carlos Sandoval

Prof. Axel Dounce

Prof. Obed Muñoz

Description of the Problem to Be Developed

In modern security systems, the integration of advanced technologies such as autonomous drones, surveillance cameras, and artificial intelligence (AI) is becoming increasingly important. These systems are designed to enhance the detection and response capabilities in high-security areas, including industrial facilities, warehouses, and public spaces. However, developing and testing such systems in real-world environments can be both costly and impractical due to safety, logistical, and ethical concerns.

This project addresses these challenges by creating a simulation environment where various security agents—including drones, surveillance cameras, and security personnel—interact to detect and respond to potential threats, such as unauthorized intrusions or thefts. The simulated environment, developed in Unity, integrates components for visual detection using the YOLOv5 deep learning model and agent-based decision-making facilitated by Python servers.

The primary problem to be tackled involves designing a coordinated system where:

1. **Surveillance Cameras** monitor the environment and identify potential threats using computer vision.
2. **Drones** respond autonomously to detected threats, verifying and tracking them while considering environmental and operational constraints such as battery levels and weather effects.
3. **Security Guards** make final decisions on whether to escalate or dismiss the identified threats, ensuring human oversight.

Agents and Their Roles

The project involves multiple agents, each with specific roles and responsibilities within the simulation. These agents work collaboratively to detect, verify, and respond to potential security threats. The key agents and their roles are as follows:

1. Surveillance Cameras (Camera Agents)

- **Role**: Monitor the environment and detect potential threats using computer vision.
- **Responsibilities**:
  - Capture images of the surroundings at regular intervals.
  - Process images through the YOLOv5 object detection model to identify objects of interest, such as intruders (e.g., a person).
  - Send alerts to other agents, such as drones or the simulation server, when a threat is detected.
  - Log system activity and detection results for further analysis.

2. Drone (Drone Agent)

- **Role**: Respond to alerts from surveillance cameras and confirm potential threats.
- **Responsibilities**:
  - Take off and patrol designated areas autonomously while maintaining optimal battery usage.
  - Receive alerts from cameras and navigate to the location of potential threats.
  - Use an onboard camera to capture and process images for threat verification.
  - Respond to environmental factors like wind and battery constraints while maintaining operational efficiency.
  - Report back to the simulation server or security guard with findings and recommendations.
  - Return to the landing station when tasks are complete or battery levels are low.

3. Security Guard (Guard Agent)

- **Role**: Provide human-like oversight and final decision-making for the system.
- **Responsibilities**:

    - Analyze drone findings and determine if detected threats are genuine or false alarms.
    - Coordinate decision-making processes by initiating a simulated voting mechanism to evaluate the severity of threats.
    - Maintain logs and oversee the system's operations to ensure optimal performance.

4. Robber (Robber Agent)

- **Role**: Simulate the presence of an intruder within the environment.
- **Responsibilities**:
    - Provide realistic scenarios for cameras and drones to detect and respond to.

5. Simulation Server

- **Role**: Act as the central controller and data repository for the simulation.
- **Responsibilities**:
    - Manage communication between agents, including sending and receiving alerts, status updates, and commands.
    - Log all interactions, metrics, and reports from agents.
    - Serve as the platform for analyzing the overall performance of the system.

6. YOLOv5 Model

- **Role**: Provide object detection capabilities for cameras and drones.
- **Responsibilities**:
    - Process images sent by the surveillance cameras and drone to identify objects of interest.
    - Return detection results, including identified objects, confidence levels, and coordinates, for further decision-making.

Relationships Between Agents

The simulation relies on a network of interactions between agents to replicate the dynamics of a real-world security system. Each agent performs specialized tasks and communicates with others to detect, verify, and respond to potential security threats. Below is an overview of the relationships between agents:

1. Surveillance Cameras ↔ Simulation Server

- **Relationship**: Surveillance cameras act as monitoring devices, continuously capturing images of the environment and sending them to the simulation server for processing.
- **Key Interactions**:
    - The camera sends detection results, including detected objects and their locations, to the server.
    - The server logs the data, evaluates the situation, and forwards alerts to relevant agents (e.g., the drone or guard).

2. Surveillance Cameras ↔ Drone

- **Relationship**: Cameras alert the drone when a potential threat is detected.
- **Key Interactions**:
    - Upon detecting a suspicious object (e.g., a person), a camera sends the drone a Call For Proposal (CFP) via the Contract Net Protocol (CNP) to investigate the identified location.
    - The drone evaluates its current status (e.g., battery level, position) and sends a proposal back to the camera, accepting or rejecting the task.

3. Drone ↔ Simulation Server

- **Relationship**: The drone serves as a mobile verification unit, updating the simulation server on its actions and findings.
- **Key Interactions**:

    - The drone sends periodic updates to the server, including battery status, position, and detection results.
    - The server logs the drone's activities and coordinates communication between the drone and other agents.

4. Drone ↔ Security Guard

- **Relationship**: The drone relies on the security guard for final decisions regarding detected threats.
- **Key Interactions**:
    - After capturing evidence, the drone notifies the guard of its findings and awaits further instructions.
    - The guard evaluates the data, conducts a voting process (if necessary), and provides directives to the drone (e.g. return to base, or dismiss the alert).

5. Security Guard ↔ Simulation Server

- **Relationship**: The security guard oversees the simulation and acts as a decision-maker, coordinating with the server for data analysis and action logging.
- **Key Interactions**:
    - The guard logs decisions and receives information about alerts and the status of other agents from the server.
    - The server serves as a communication hub, ensuring the guard has the necessary data to make informed decisions.

6. Robber ↔ Surveillance Cameras and Drone

- **Relationship**: The robber serves as a test subject for the surveillance cameras and drone, simulating an intruder.

7. YOLOv5 Model ↔ Surveillance Cameras and Drone

- **Relationship**: YOLOv5 acts as the detection engine, assisting cameras and the drone in identifying potential threats.
- **Key Interactions**:
  - Cameras and drones send image data to the YOLOv5 server for processing.
  - YOLOv5 returns detection results, including detected objects, confidence levels, and coordinates, enabling further decision-making by the respective agents.

8. Robber ↔ Security Guard

- **Relationship**: The robber indirectly interacts with the security guard by influencing decisions.
- **Key Interactions**:
  - The guard analyzes the drone's findings regarding the robber and decides whether to escalate or dismiss the threat.

Agent Properties with Justification for Each Property (Based on Provided Scripts)

1. Surveillance Cameras (Camera Agents)

- **Properties:**
  - **surveillanceCamera** **(Camera):** References the camera component of the object. This property is essential for capturing images of the simulation environment.
  - **cameraId** **(String):** A unique identifier for each camera. This allows the simulation server and other agents to identify the source of detection data.
  - **serverUrl** **(String):** Specifies the URL for the YOLOv5 server. Enables communication with the detection model for image processing.
  - **captureInterval** **(Float):** Defines the time interval (in seconds) between image captures. Helps balance detection frequency and computational resource usage.

- **captureWidth and captureHeight (Integers):** Specify the resolution of the captured images. Higher resolution improves detection accuracy but requires more processing power.
- **thiefDetected (Static Boolean):** Tracks whether a thief has been detected by any camera. Shared across all camera agents to coordinate their behavior.
- **totalImagesSent (Integer):** Counts the total number of images sent to the YOLOv5 server. Helps measure performance and activity levels.
- **successfulDetections (Integer):** Tracks the number of successful detections. Used for evaluating the system's effectiveness.
- **totalLatency and latencySamples (Floats and Integers):** Measure the time taken for the YOLOv5 server to process images and respond. Crucial for analyzing system responsiveness.

- **Justification:** These properties ensure that each camera agent can monitor the environment effectively, detect threats, and communicate findings to the rest of the system.

2. Drone (Drone Agent)

- **Properties:**
  - **landingStationPosition (Vector3):** Specifies the position of the drone's base. Used for returning and landing operations.
  - **cameraDetectionTime (Float):** Stores the time at which a camera detected a thief. Helps calculate response times.
  - **currentBatteryLevel (Float):** Tracks the drone's battery level. Enables decisions such as returning to base when the battery is low.
  - **takeOffHeight and moveSpeed (Floats):** Define the altitude for patrolling and the movement speed of the drone. These parameters control the drone's navigation.
  - **patrolPoints (List of Vector3):** Contains predefined waypoints for patrolling. Enables the drone to autonomously cover specific areas.

    - **windStrength (Float):** Represents the effect of wind on the drone's movement. Adds realism to the simulation by simulating environmental challenges.
    - **thiefDetected (Boolean):** Tracks whether the drone has detected a thief. Coordinates with cameras and the guard to confirm threats.
    - **detectionConfirmationTimes (List of Floats):** Stores the times taken to confirm detections. Useful for evaluating system performance.
    - **droneTotalLatency and droneLatencySamples (Floats and Integers):** Measure the drone’s communication efficiency with the YOLOv5 server.
- **Justification:** These properties enable the drone to operate autonomously, perform threat verification, and adapt to environmental and operational constraints.

3. Security Guard (Guard Agent)

- **Properties:**
    - **status (String):** Represents the guard’s operational state (e.g., "Idle", "Analyzing"). Tracks the agent's role in decision-making processes.
    - **messageQueue (Simulated Messages):** Used for receiving and processing alerts or votes. Coordinates with drones and cameras.
    - **voteResults (Boolean Decision):** Simulates the result of a voting process to assess threats. Adds an element of human-like decision-making to the system.
- **Justification:** These properties allow the guard to act as a human decision-maker, ensuring threats are analyzed thoroughly and false alarms are minimized.

4. Robber (Robber Agent)

- **Properties:**
    - **position (Vector3):** Specifies the current location of the robber. Allows cameras and drones to locate and respond to the intruder.

    - **status (String):** Tracks the robber’s activity (e.g., "Active", "Idle"). Adds variability to the simulation by simulating different behaviors.
    - **simulationEffectiveness (Implicit):** The robber’s presence indirectly measures the system’s ability to detect and respond to threats.
- **Justification:** These properties make the robber a dynamic element in the simulation, providing realistic scenarios to test the security system's capabilities.

5. Simulation Server

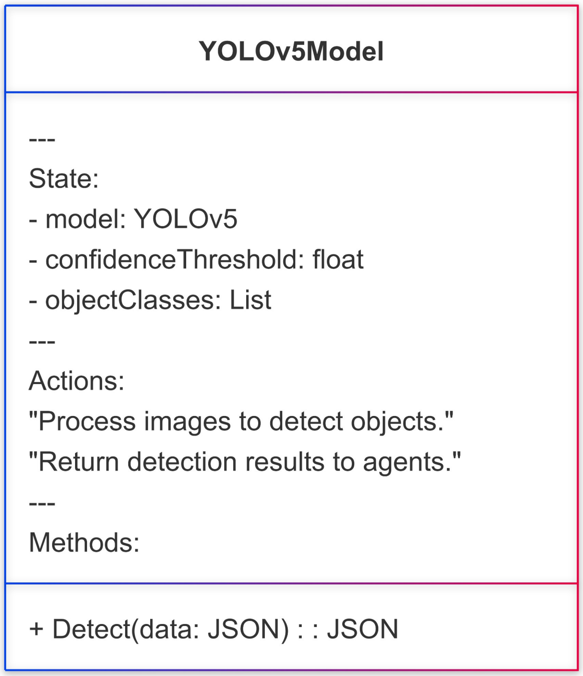
- **Properties:**
    - **agentRegistry (Dictionary):** Tracks all agents, including their properties and states. Centralizes management and coordination.
    - **logSystem (List of Strings):** Records interactions and decisions. Enables debugging and performance analysis.
    - **performanceMetrics (Dictionaries):** Captures key metrics like detection rates, latency, and battery usage. Supports system evaluation and optimization.
    - **messageQueue (Queue):** Manages inter-agent communication. Facilitates smooth and asynchronous operations within the simulation.
- **Justification:** These properties allow the simulation server to act as the central communication hub, ensuring seamless integration and comprehensive data analysis.

6. YOLOv5 Model

- **Properties:**
    - **model (Pretrained YOLOv5):** The detection model used for identifying objects in images. Essential for accurate threat detection.
    - **confidenceThreshold (Float):** The minimum confidence required for a detection to be considered valid. Balances precision and recall.
    - **objectClasses (List of Strings):** Specifies the classes of objects the model can detect (e.g., "person"). Aligns with the system's objectives.

- **Justification:** These properties define the YOLOv5 model's capabilities, ensuring it provides accurate and reliable object detection for cameras and drones.

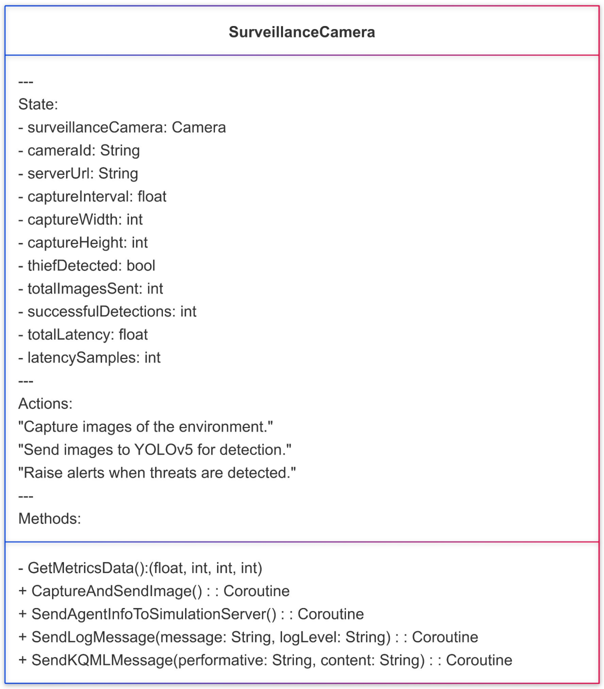
Agent class diagrams


YOLOv5Model

State:
- model: YOLOv5
- confidenceThreshold: float
- objectClasses: List

Actions:
"Process images to detect objects."
"Return detection results to agents."

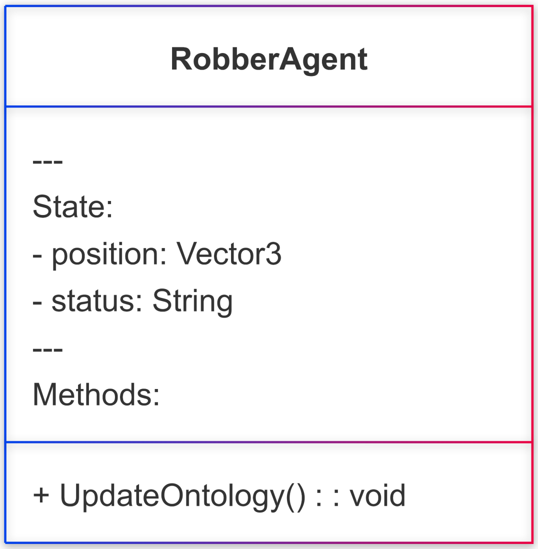
Methods:
+ Detect(data: JSON) : : JSON

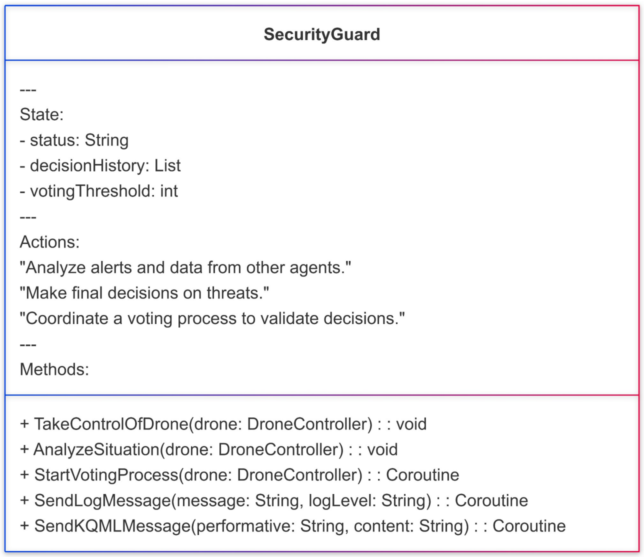

SurveillanceCamera

State:
- surveillanceCamera: Camera
- cameraId: String
- serverUrl: String
- captureInterval: float
- captureWidth: int
- captureHeight: int
- thiefDetected: bool
- totalImagesSent: int
- successfulDetections: int
- totalLatency: float
- latencySamples: int

Actions:
"Capture images of the environment."
"Send images to YOLOv5 for detection."
"Raise alerts when threats are detected."

Methods:
- GetMetricsData():(float, int, int, int)
+ CaptureAndSendImage() : : Coroutine
+ SendAgentInfoToSimulationServer() : : Coroutine
+ SendLogMessage(message: String, logLevel: String) : : Coroutine
+ SendKQMLMessage(performative: String, content: String) : : Coroutine

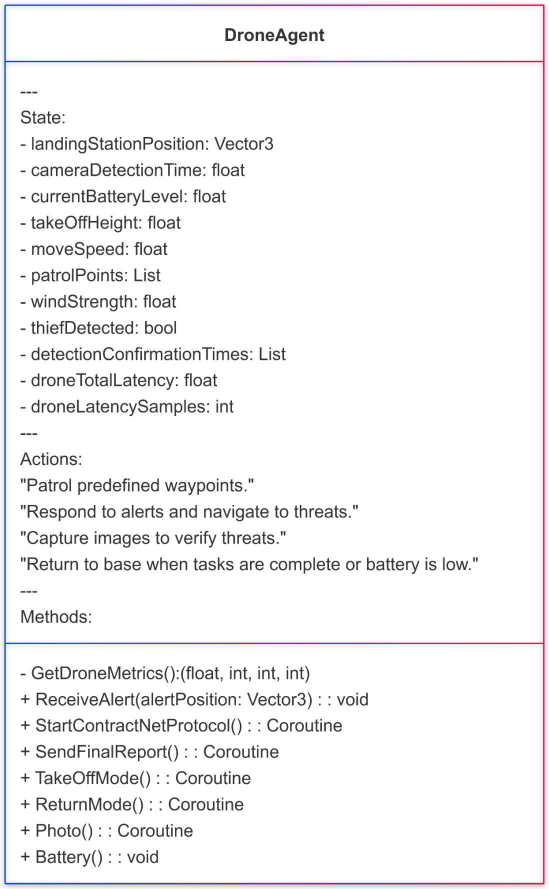



SecurityGuard

State:
- status: String
- decisionHistory: List
- votingThreshold: int

Actions:
"Analyze alerts and data from other agents."
"Make final decisions on threats."
"Coordinate a voting process to validate decisions."

Methods:
+ TakeControlOfDrone(drone: DroneController) : : void
+ AnalyzeSituation(drone: DroneController) : : void
+ StartVotingProcess(drone: DroneController) : : Coroutine
+ SendLogMessage(message: String, logLevel: String) : : Coroutine
+ SendKQMLMessage(performative: String, content: String) : : Coroutine



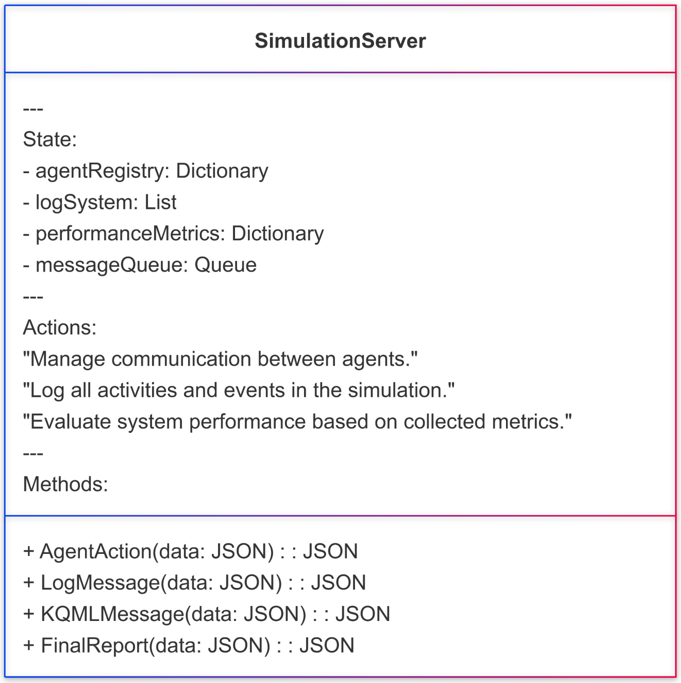
DroneAgent

State:
- landingStationPosition: Vector3
- cameraDetectionTime: float
- currentBatteryLevel: float
- takeOffHeight: float
- moveSpeed: float
- patrolPoints: List
- windStrength: float
- thiefDetected: bool
- detectionConfirmationTimes: List
- droneTotalLatency: float
- droneLatencySamples: int

Actions:
"Patrol predefined waypoints."
"Respond to alerts and navigate to threats."
"Capture images to verify threats."
"Return to base when tasks are complete or battery is low."

Methods:

- GetDroneMetrics():(float, int, int, int)
+ ReceiveAlert(alertPosition: Vector3) : : void
+ StartContractNetProtocol() : : Coroutine
+ SendFinalReport() : : Coroutine
+ TakeOffMode() : : Coroutine
+ ReturnMode() : : Coroutine
+ Photo() : : Coroutine
+ Battery() : : void



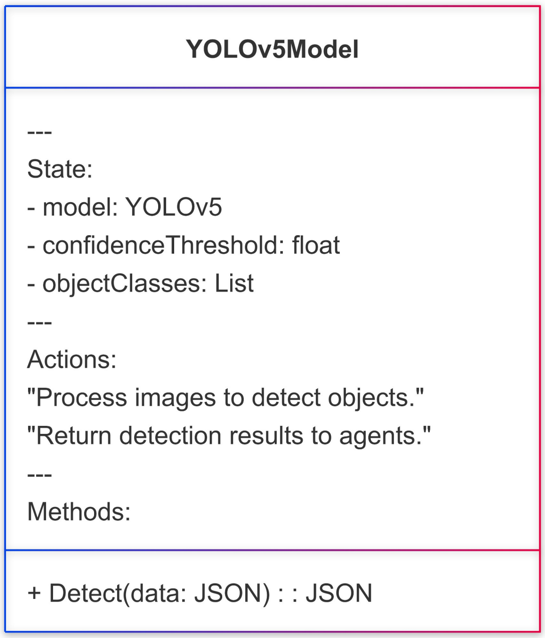
SimulationServer

State:
- agentRegistry: Dictionary
- logSystem: List
- performanceMetrics: Dictionary
- messageQueue: Queue

Actions:
"Manage communication between agents."
"Log all activities and events in the simulation."
"Evaluate system performance based on collected metrics."

Methods:

+ AgentAction(data: JSON) : : JSON
+ LogMessage(data: JSON) : : JSON
+ KQMLMessage(data: JSON) : : JSON
+ FinalReport(data: JSON) : : JSON



YOLOv5Model

State:
- model: YOLOv5
- confidenceThreshold: float
- objectClasses: List

Actions:
"Process images to detect objects."
"Return detection results to agents."

Methods:

+ Detect(data: JSON) : : JSON

Metrics and Success Measurement of the Simulation

The success of the simulation is assessed through quantitative metrics derived from agent interactions, system performance, and overall outcomes. Below is an outline of key metrics, their interpretations, and graphs to visualize the simulation's performance.

Key Metrics

1. **Detection Accuracy:**

   - **Definition:** The percentage of successful detections (true positives) relative to the total images processed by cameras and drones.
   - **Calculation:** Detection Accuracy (%)=(Successful DetectionsTotal Images Processed×100)\text{Detection Accuracy (\%)} = \left(\frac{\text{Successful Detections}}{\text{Total Images Processed}} \times 100\right)

2. **Average Latency:**

   - **Definition:** The average time taken by the YOLOv5 model to process images and return results.
   - **Calculation:** Average Latency (s)=Total LatencyLatency Samples\text{Average Latency (s)} = \frac{\text{Total Latency}}{\text{Latency Samples}}

3. **Battery Efficiency:**

   - **Definition:** The average battery consumption per patrol cycle for the drone.
   - **Calculation:** Battery Consumption per Cycle (%)=Total Battery ConsumedCompleted Patrol Cycles\text{Battery Consumption per Cycle (\%)} = \frac{\text{Total Battery Consumed}}{\text{Completed Patrol Cycles}}

4. **Time-to-Confirmation:**

   - **Definition:** The time between initial detection of a threat by cameras and confirmation by the drone or security guard.
   - **Interpretation:** Lower values indicate efficient coordination among agents.

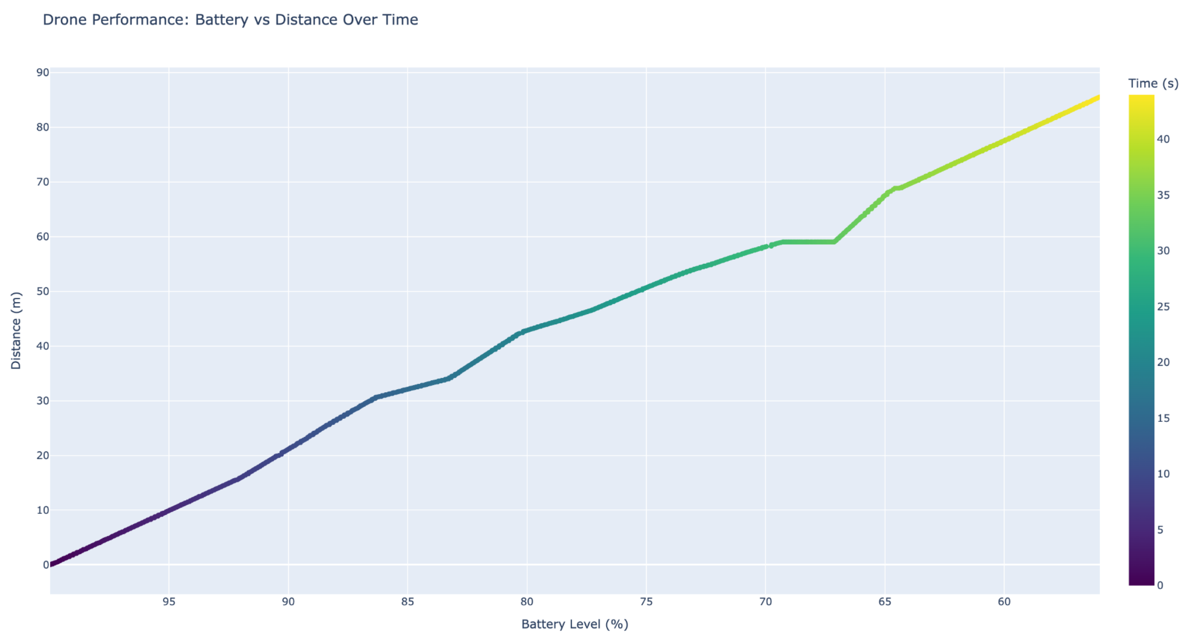
5. **Distance Traveled by Drone:**

   - **Definition:** Total distance traveled by the drone during the simulation.
   - **Calculation:** Sum of distances moved between consecutive positions.

6. **False Alarms:**

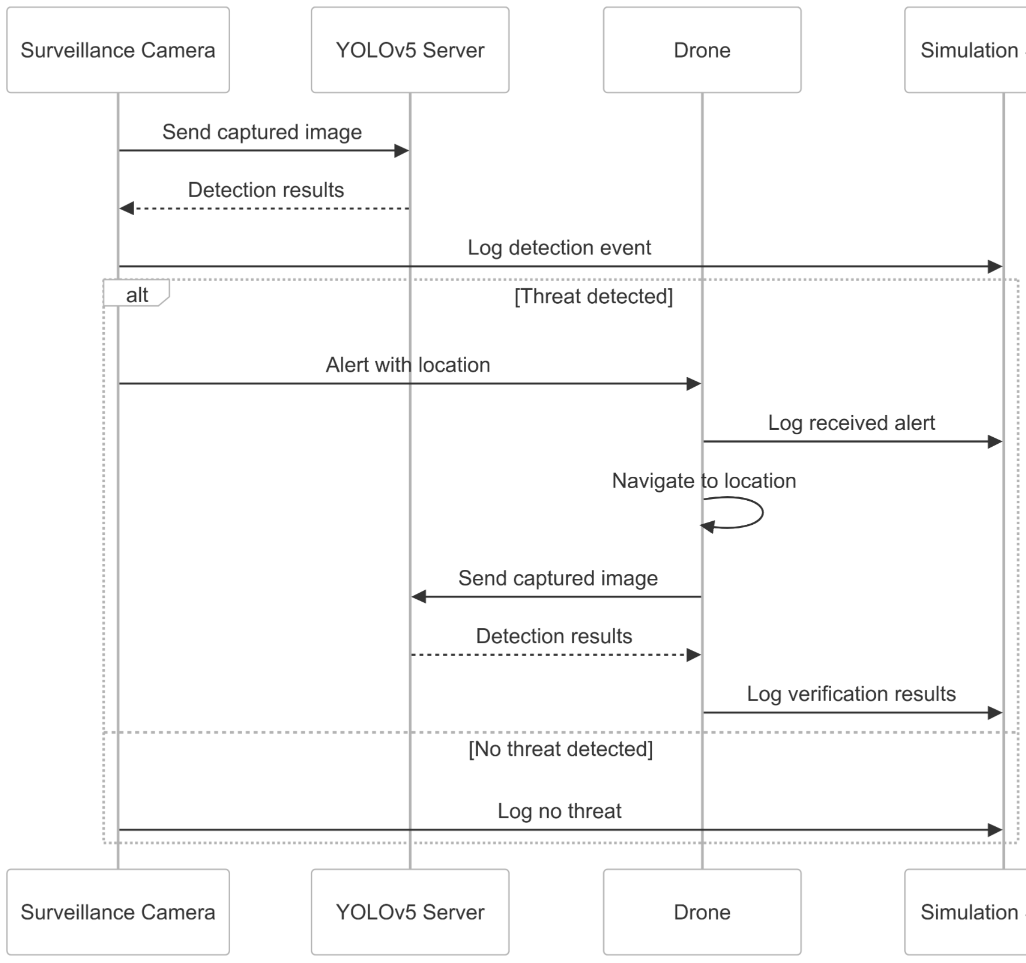
   - **Definition:** The number of detections classified as threats that were later dismissed as false alarms by the security guard.
   - **Interpretation:** Lower false alarm rates indicate better system reliability.

Drone Performance: Battery vs Distance Over Time

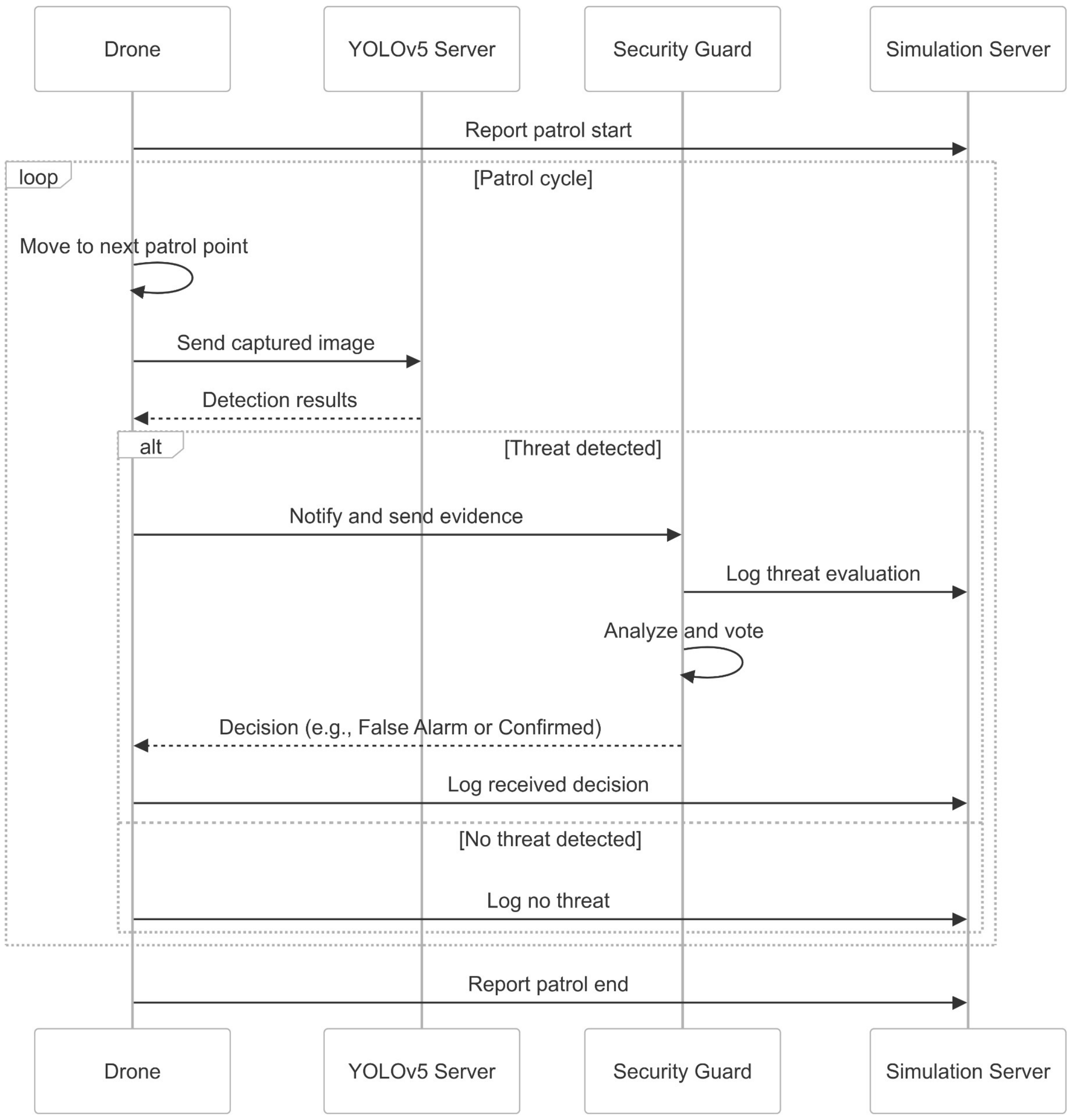


Sequence diagrams of interaction protocols

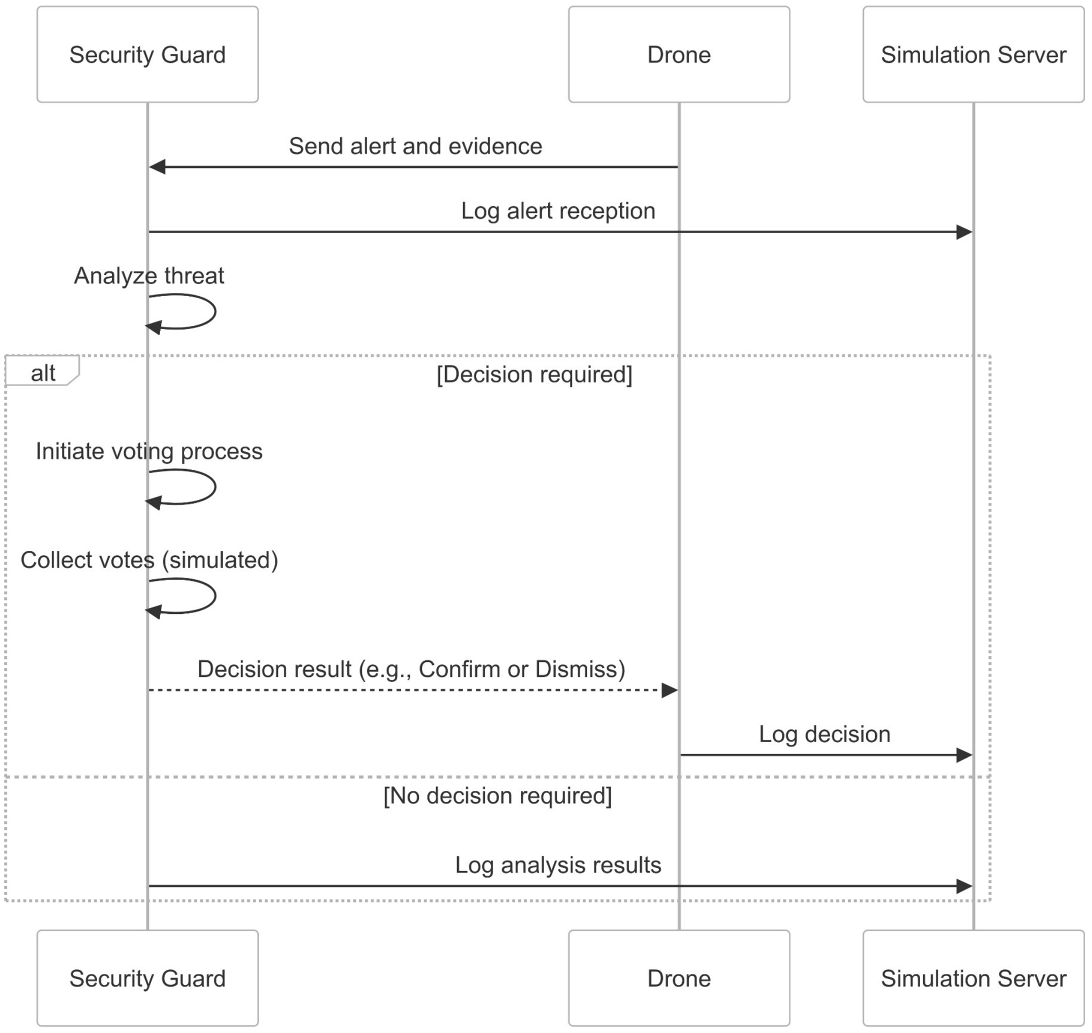
1. Detection and Alert Protocol



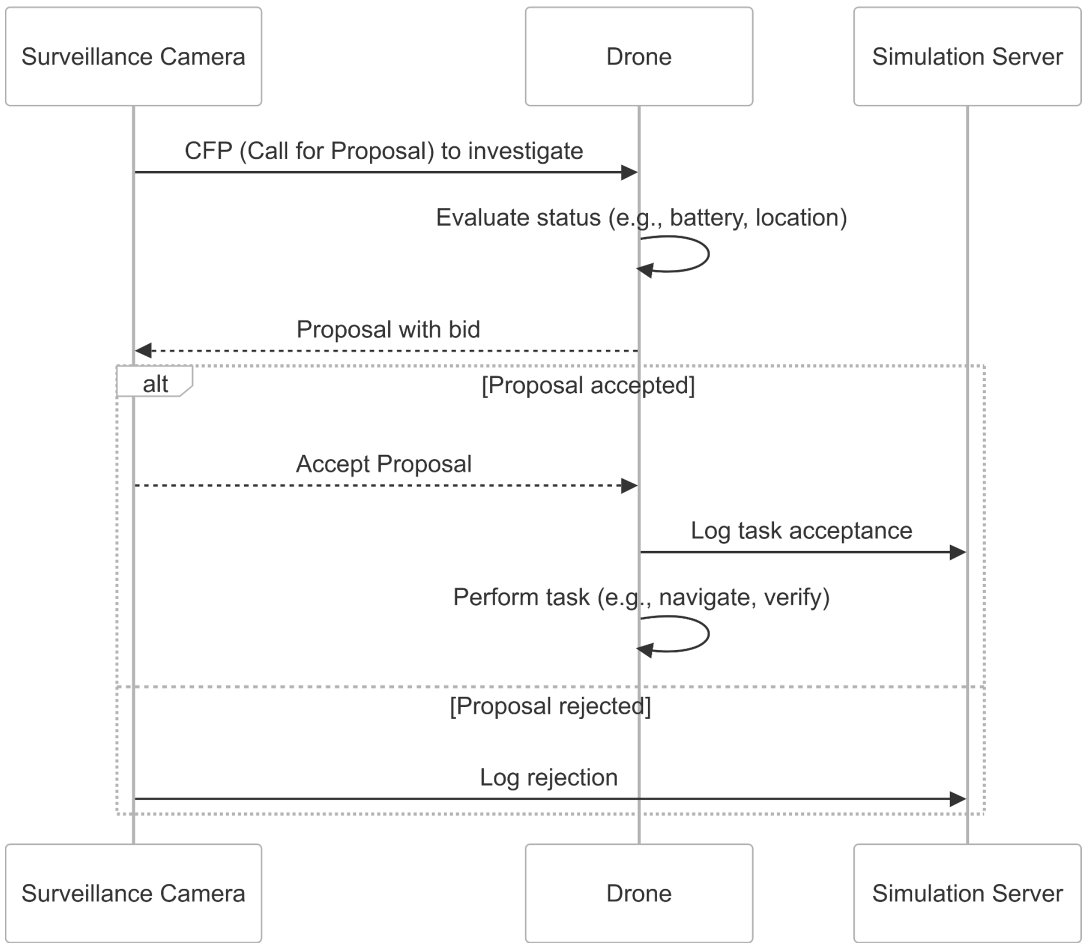
2. Drone Patrol and Threat Verification



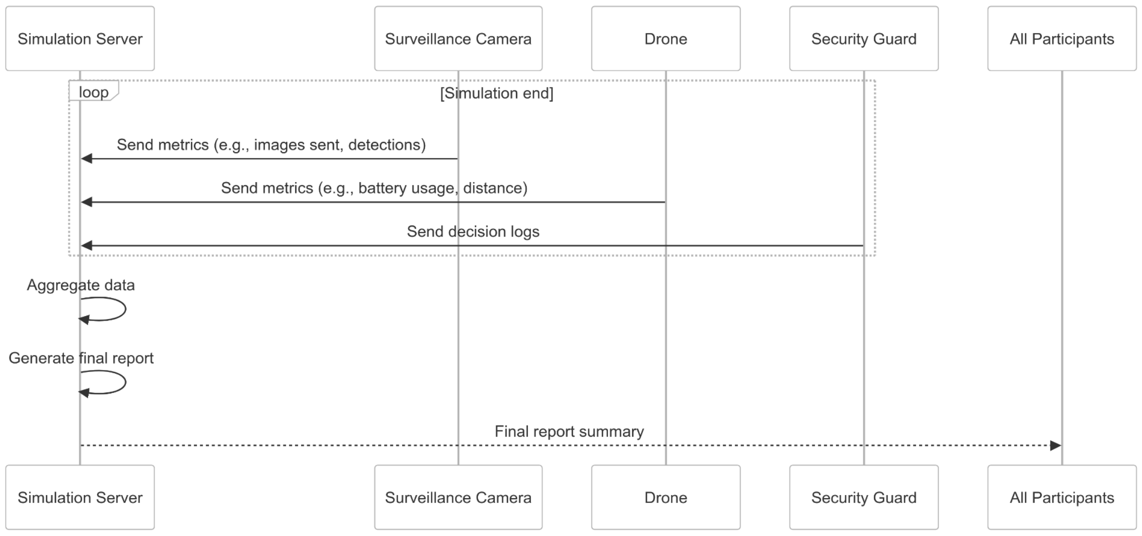
3. Security Guard Decision-Making Protocol



4. Contract Net Protocol for Task Assignment



5. Final Report Generation



Analysis of the Developed Solution

1. **Why did you select the multi-agent model used?**

   We selected the multi-agent model because it closely resembles actual security systems, where all agents work collaboratively to detect, verify, and respond to potential threats. The model offers exceptional modularity, allowing for easy expansion and specialized task allocation. By creating agents with distinct roles and autonomous capabilities, we could simulate a complex security environment that mimics real-world decision-making processes.

2. **What were the variables that were taken into account at the time of making the decision?**

   Five primary variables guided our decision-making: agent specialization, communication efficiency, environmental constraints, threat complexity, and system modularity. Agent specialization ensured each component had a clear, optimized role. Communication efficiency enabled real-time interactions between agents. Environmental constraints like battery levels and wind effects added realism. Threat complexity demanded system flexibility, while modularity allowed for future expansions and adaptations.

3. **What is the interaction of these variables with respect to the simulation result?**

   These variables dynamically interact to shape the simulation's behavior. For instance, the drone's battery level directly impacts its response capabilities, while the YOLOv5 model's detection accuracy influences camera performance. Communication protocols determine alert transmission speed, and agent specialization ensures optimal task allocation. This intricate interaction allows the system to adapt to changing scenarios, simulate realistic security responses, and provide comprehensive threat detection and verification.

4. **Why did you select the graphical design presented?**

We chose the design to represent a realistic location where theft identification would typically be critical - an area with containers where an intruder might attempt to steal contents. This design enables a realistic simulation of potential security challenges, provides a comprehensible environment for visualizing agent interactions, and demonstrates how different agents detect and respond to threats in a tangible, relatable context.

5. **What are the advantages you find in the final solution presented?**

   The solution offers several key advantages. The Unity-based scene provides high realism, while the multi-agent model ensures modularity and easy expandability. Agents operate autonomously yet collaborate effectively, creating a dynamic security system. The use of Python facilitates comprehensive data analysis and graphing. The implementation allows for autonomous operation of components like the drone, and the collaborative approach between agents ensures more robust threat detection and response.

6. **What are the disadvantages that exist in the solution presented?**

   Several disadvantages emerged during development. Communication between Python and Unity servers can introduce latency issues. The multi-agent system's complexity makes debugging challenging. Despite the simulation's realism, it may not fully capture all unexpected scenarios that could occur in a real-world security environment. The current implementation, while sophisticated, still represents a simplified model of potential security challenges.

7. **What modifications could you make to reduce or eliminate the disadvantages mentioned?**

To address these limitations, we propose several modifications. Integrating the entire framework within a single environment could minimize latency errors. Implementing a coordinating meta-agent could help manage and reduce system complexity. Introducing more variables that might occur in real-world scenarios would enhance the simulation's realism. Additionally, developing more sophisticated communication protocols and expanding the environmental constraints could provide a more comprehensive security simulation.

Reflection on the Learning Process

This project was an invaluable learning experience, providing deep insights into multi-agent systems, real-time simulations, and AI integration. We gained a comprehensive understanding of agent-based modeling, learned to balance simulation complexity with computational feasibility, and developed advanced problem-solving skills. The iterative development process taught us the importance of continuous testing, refinement, and collaboration. Most importantly, we learned to approach complex technological challenges with persistence, creativity, and a systematic mindset.

Link for Github Repository

https://github.com/A01644090/Evidence-2-Multiagent-Systems