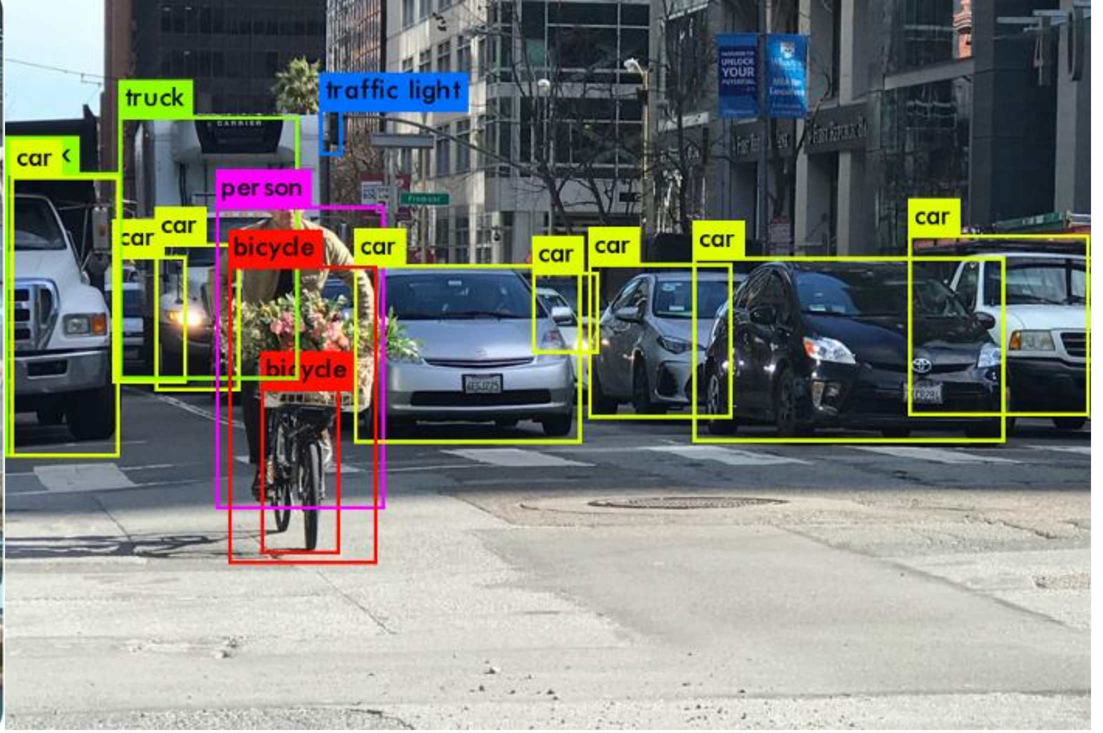
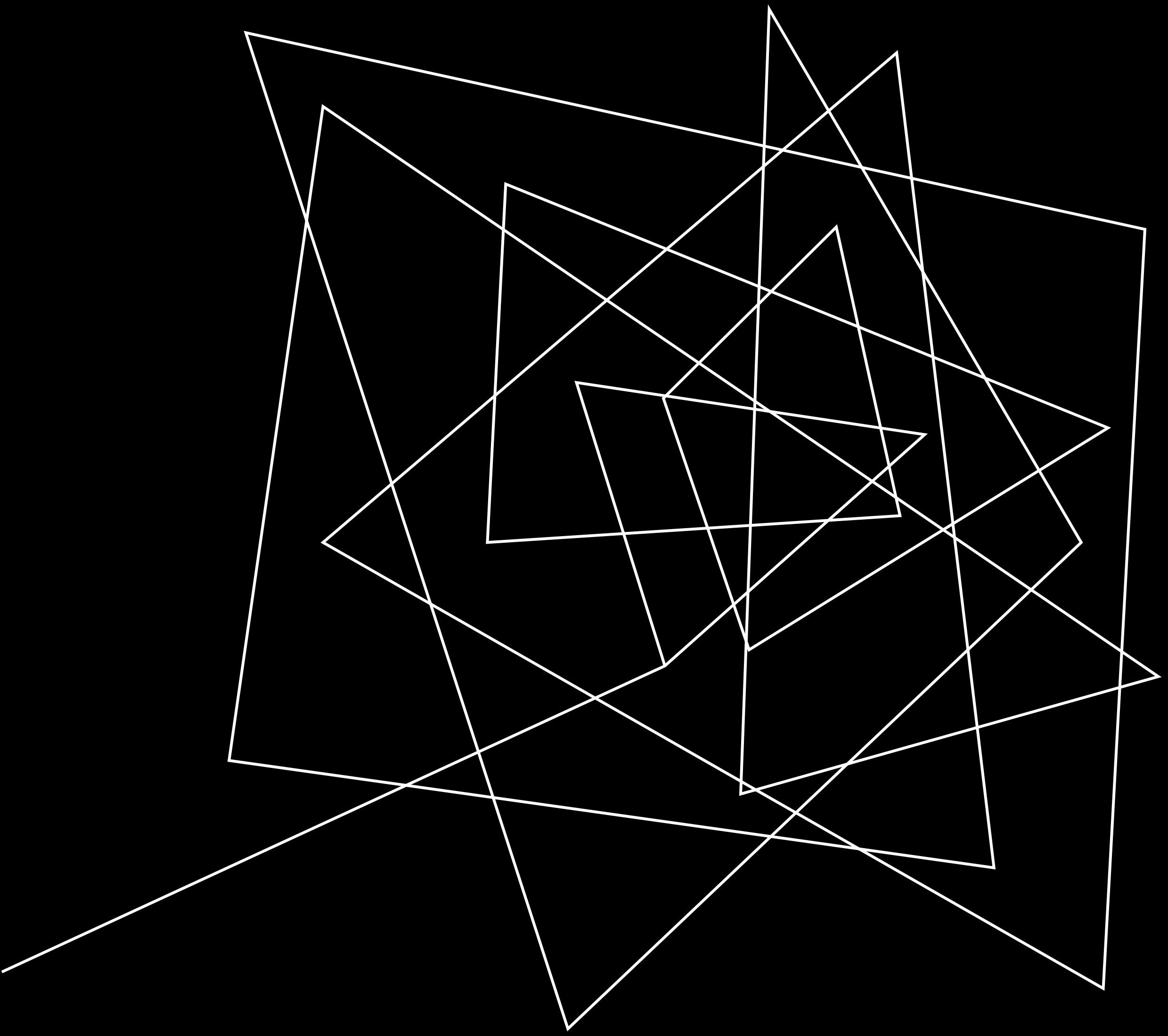


- **PROJECT TITLE:** AI-ENABLED DRONE SWARM NETWORK AND INFORMATION MANAGEMENT
  SUBTITLE: OBJECT DETECTION, DRONE2DRONE INFORMATION EXCHANGE AND COLLABORATIVE AWARENESS



MQTT

COMMUNICATION FOR COLLABORATIVE DRONES

- Swarm-based drone operations
- How drones can share navigation data
- Key communication challenges: latency, reliability, scalability
- Solution: Lightweight MQTT protocol with Mosquitto broker

MQTT PROTOCOL BASICS

Understanding MQTT for Drone Communication

1. MQTT (Message Queuing Telemetry Transport)
    1. Lightweight, low-bandwidth, real-time messaging
    2. Publisher-Subscriber model
    3. QoS levels: 0 (At most once), 1 (At least once), 2 (Exactly once)
2. Mosquitto as an MQTT broker

SYSTEM ARCHITECTURE

Drone Swarm Communication Architecture

1. Components:Drones with MQTT clients
    - Mosquitto broker (central or distributed)
    - Object detection module
2. Communication flow:
    - Drone detects obstacle
    - Publishes obstacle data to MQTT topic
    - Other drones subscribe and adjust navigation

SETTING UP MQTT WITH MOSQUITTO

Installing Mosquitto MQTT Broker

1. Install Mosquitto:

```
sudo apt update
sudo apt install mosquitto mosquitto-clients
```

2. Start the Mosquitto broker:

```
sudo systemctl start mosquito
```

3. Test publishing and subscribing:

```
mosquitto_sub -h localhost -t "drone/obstacles“
```

```
mosquitto_pub -h localhost -t "drone/obstacles" -m "Obstacle: Tree, Position: x,y, Height: z, Not moving"
```

MQTT TOPICS AND MESSAGE FORMAT

Defining Topics and Payload Structure

Example topic structure:

drone/obstacles
drone/navigation

JSON-based message format:


```
{
    "type": "obstacle",
    "object": "tree",
    "position": {"x": 25, "y": 30},
    "height": 15,
    "movement": "stationary“
}
```

IMPLEMENTING MQTT IN DRONES

Integrating MQTT with Drone Software - Example

- Install MQTT Python client:

```
pip install paho-mqtt
```

- Implement a drone publisher:

```
import paho.mqtt.client as mqtt
import json
client = mqtt.Client()
client.connect("localhost", 1883, 60)

obstacle_data = {
"type": "obstacle",
"object": "tree",
"position": {"x": 25, "y": 30},
"height": 15,
"movement": "stationary“
}
client.publish("drone/obstacles", json.dumps(obstacle_data))
client.disconnect()
```

HANDLING MESSAGES IN SUBSCRIBER DRONES

Receiving and Processing Obstacle Data

1. Implement a drone subscriber:

```
def on_message(client, userdata, msg):
data = json.loads(msg.payload)
print(f"Received obstacle: {data}")
# Adjust navigation accordingly

client = mqtt.Client()
client.on_message = on_message
client.connect("localhost", 1883, 60)
client.subscribe("drone/obstacles")
client.loop_forever()
```

ENHANCING COMMUNICATION

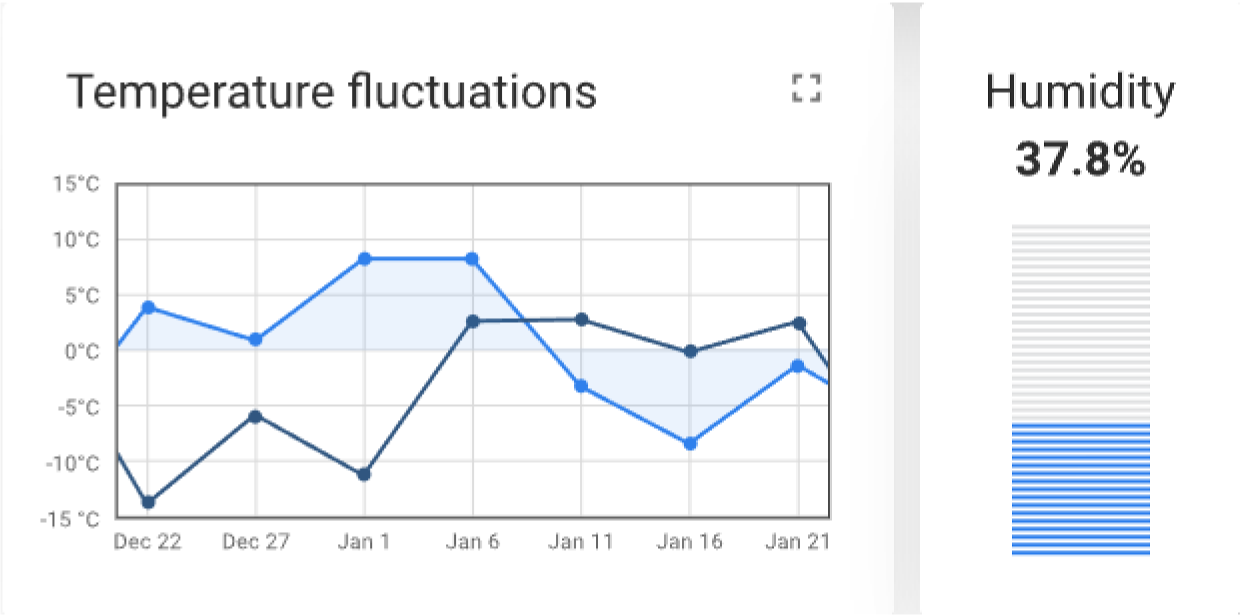
Improving the System…

- Implement QoS for reliable delivery
- Use distributed MQTT brokers for scalability
- Encrypt messages using TLS for security
- Integrate GPS and real-time AI / smallLLM processing

THINGSBOARD (1)

ThingsBoard CE - https://thingsboard.io/

- Open-source IoT Platform
- Device management, data collection, processing and visualization for your IoT solution (drones in our case)



THINGSBOARD (2)

- Create custom sensors to store and present drone-related data (battery level, status, availability for detection, payload)
- Sensors' values should be updated from a dataset or randomly periodically.
- Drones should be able to send their data to Thingsboard. Drones and LLM should be able to retrieve data from Thingsboard.

THINGSBOARD – INSTALLATION (2)

https://thingsboard.io/docs/user-guide/install/ubuntu/

https://thingsboard.io/docs/user-guide/install/windows/

Follow the guides above and choose the following:

For the Database:

PostgreSQL
(recommended for < 5K msg/sec)

For the queue service:

In Memory
(built-in and default)

You are free to try other options but the recommended choices are the simplest ones.

THINGSBOARD – USAGE (1)

https://thingsboard.io/docs/user-guide/install/ubuntu/

https://thingsboard.io/docs/user-guide/install/windows/

- Getting started guides - These guides provide quick overview of main ThingsBoard features. Designed to be completed in 15-30 minutes.
- Connect your device - Learn how to connect devices based on your connectivity technology or solution.
- Data visualization - These guides contain instructions how to configure complex ThingsBoard dashboards.
- Data processing & actions - Learn how to use ThingsBoard Rule Engine.
- IoT Data analytics - Learn how to use rule engine to perform basic analytics tasks.
- Hardware samples - Learn how to connect various hardware platforms to ThingsBoard.
- Advanced features - Learn about advanced ThingsBoard features.

THINGSBOARD – USAGE (2)

Hello world

Learn how to collect IoT device data using MQTT, HTTP or CoAP and visualize it on a simple dashboard. Provides variety of sample scripts that you can run on your PC or laptop to simulate the device.

End user IoT dashboards

Learn how to perform basic operations over Devices, Customers, and Dashboards.

Device data management

Learn how to perform basic operations over device attributes to implement practical device management use cases.

Getting started with Rule Engine

Learn about ThingsBoard rule engine and typical use cases you can implement. Review Hello World example and learn how-to enable filtering of incoming telemetry messages.

INTERDRONE COMMUNICATION WITH LLM

PREREQUISITES

LLM-Based Inter-Drone Communication

- Step 1: Choosing the LLM Model

- Compact and efficient models are preferred for edge inference.
- Options:
- LLaMA 2 (7B): Small enough for fine-tuning while maintaining strong reasoning abilities.
- LLaMA 3 (8GB): Small enough for fine-tuning while maintaining strong reasoning abilities.
- GPT-NeoX-20B: Larger, but may require cloud-based inference.
- Mistral-7B: A newer, efficient alternative.

- We propose LLaMA 2/3 since it is open-source and provides a balance of efficiency and accuracy.

There are many guides online for running ollama locally with WebUI but an example could be:
https://dev.to/timesurgelabs/how-to-run-llama-3-locally-with-ollama-and-open-webui-297d

FINE-TUNING THE LLM FOR DRONE COMMUNICATION

Objective: Train the LLM on structured drone communications to understand and generate meaningful responses.

Dataset Requirements:

- Drone telemetry logs.
- Example drone dialogues.
- Mission objectives and collaborative decision-making samples.

EXAMPLE TRAINING DATA FORMAT (JSONL)

```
{"input": "Drone 1: Obstacle detected at (15m, North). Suggested action?", "output": "Adjust course 5 degrees right to avoid the obstacle."}
{"input": "Drone 2: Low battery warning. How should I proceed?", "output": "Return to base if below 20% charge; otherwise, continue with mission."}
{"input": "Drone 3: Clear path detected. Continue mission?", "output": "Affirmative. Proceed along planned route."}
```

FINE-TUNING THE LLM FOR DRONE COMMUNICATION

Fine-tuning Approach:

- Using LLaMA 2: Use QLoRA or LoRA for lightweight fine-tuning.
- Steps:
  - Load dataset in Hugging Face format.
  - Apply LoRA for efficient fine-tuning.
  - Save the fine-tuned model for deployment.

IMPLEMENTING MESSAGE ENCODING AND DECODING

The LLM will convert numerical sensor data into human-readable contextual messages.
Example: Raw Drone Data


```
{
  "drone_id": "Drone_1",
  "sensor_data": {
    "obstacle_distance": 15,
    "direction": "North"
  },
  "command": "suggest_action"
}
```


Encoded Message with LLM


```
{
  "message": "Obstacle detected: Tree at 15m ahead. Suggested course adjustment: 5 degrees right."
}
```

IMPLEMENTING CONTEXTUAL DECISION-MAKING

Maintaining Conversation History: Drones need memory of past exchanges.
Approach: Use Sliding Context Window (last 3 messages).
Implementation: Store past messages and append them for input processing.

```
class DroneLLM:
    def __init__(self, model_path):
        self.tokenizer = AutoTokenizer.from_pretrained(model_path)
        self.model = AutoModelForCausalLM.from_pretrained(model_path)
        self.history = []  # Store last N messages

    def process_message(self, new_input):
        # Append new message to history
        self.history.append(new_input)
        if len(self.history) > 3:  # Keep only last 3 messages
            self.history.pop(0)
```

FINE-TUNING THE MODEL - OPTIMIZING FOR BETTER RESULTS

```
    # Create prompt for LLM
    context = " ".join(self.history)
    input_text = f"Context: {context}\nDrone: {new_input}\nLLM Response:"

    # Generate response
    inputs = self.tokenizer(input_text, return_tensors="pt")
    output = self.model.generate(**inputs, max_length=100)
    response = self.tokenizer.decode(output[0], skip_special_tokens=True)

    return response
```

IMPLEMENTING LLAMA 2 FOR INTER-DRONE COMMUNICATION

Requirements:

- Hugging Face Transformers
- LoRA for fine-tuning
- MQTT for communication

IMPLEMENTING LLAMA 2 FOR INTER-DRONE COMMUNICATION

Load LLaMA 2 Model

```
from transformers import AutoModelForCausalLM, AutoTokenizer
import torch

# Load LLaMA 2 (7B)
model_path = "meta-llama/Llama-2-7b-chat-hf"
tokenizer = AutoTokenizer.from_pretrained(model_path)
model = AutoModelForCausalLM.from_pretrained(model_path, device_map="auto")
```

IMPLEMENT MESSAGE PROCESSING

```
def generate_drone_response(input_message):
    prompt = f"Drone: {input_message}\nLLM Response:"

    inputs = tokenizer(prompt, return_tensors="pt").to("cuda")
    outputs = model.generate(**inputs, max_length=100)
    response = tokenizer.decode(outputs[0], skip_special_tokens=True)

    return response
```

IMPLEMENT MESSAGE PROCESSING

Example:

```
message = "Obstacle detected at 15m. What should I do?"
response = generate_drone_response(message)
print(response)
```

Expected Output:

"Adjust course 5 degrees right to avoid the obstacle."

REAL-TIME COMMUNICATION VIA MQTT

To integrate drones, we use MQTT for real-time message exchange.

3.1 Install MQTT Library

```
pip install paho-mqtt
```

3.2 Drone LLM Publisher

```
import paho.mqtt.client as mqtt
import json
broker = "mqtt.eclipseprojects.io"
topic = "drones/llm"
client = mqtt.Client()
client.connect(broker)
message = {
    "drone_id": "Drone_1",
    "message": "Obstacle detected at 15m. What should I do?"
}
client.publish(topic, json.dumps(message))
print("Sent message:", message)
client.disconnect()
```

DRONE LLM SUBSCRIBER

```
def on_message(client, userdata, msg):
    data = json.loads(msg.payload)
    print(f"Received from {data['drone_id']}: {data['message']}")

    response = generate_drone_response(data['message'])
    print("LLM Response:", response)

client = mqtt.Client()
client.connect(broker)
client.subscribe(topic)
client.on_message = on_message
client.loop_forever()
```

ΔΙΑΔΙΚΑΣΤΙΚΑ ΜΑΘΗΜΑΤΟΣ ΟΜΑΔΩΝ MQTT + LLM

- 1 άτομο της ομάδας MQTT θα συνεργαστεί με 1 άτομο της ομάδας LLM
- Θα βαθμολογηθεί η συμμετοχή στο μάθημα, όχι μόνο το τελικό αποτέλεσμα