Intelligent Boat Patrolling System

Redes e Sistemas Autónomos - 2023

Nuno Fahla (97631), Miguel Tavares (98448)

Table of Contents

Introduction

Maritime patrols are vital for maintaining the safety and security of maritime environments, however, this poses a threat to human safety and the equipment in use. Making use of software that is easy to customize and expand, purpose-built, cost-effective and easy to deploy, we want to automate these maritime patrols through Vehicle-to-Vehicle communication to reliably explore surrounding maritime areas, while ensuring collaborative positioning, path avoidance and location sharing.

Objectives

Our main goal is to develop an autonomous patrol system using boats in a maritime area to gather relevant information and scout the whole area as fast and as efficiently as possible. The maritime area is divided into a grid with reference points which must be scouted, after that, the boats must form an Ad Hoc network with each other to carry out the task independently of the number of starting boats, as long as at least one boat survives. Between the active nodes, the whole area must be patrolled, passing through previously patrolled areas as little as possible and avoiding collisions. To easily coordinate their movement and paths, the boats should communicate with each other about the reference points they have discovered. All the boats should relay their current location to the tracking platform, in real time. These communications must adhere to the standards outlined in the ETSI C-ITS protocol suite, ensured by the NAP-Vanetza which acts as a wrapper to allow MQTT messages in the JSON format.

Architecture

Having built our system to be as scalable as possible, we can't provide a fixed architecture diagram, since we can have a varying amount of Boat nodes, *figure 1* below exemplifies our for 3 boats.

Communication between all elements in our network is confined to a Docker Network, ensuring boats communicated between themselves and to the tracking platform.

Our tracking platform is based on a Flask API, with only one endpoint, which accepts *GET* requests made by *AJAX* calls in our *JavaScript* file, to know where to represent the latest boat positions in the *Leaflet* Map. This API doesn't need a *POST* endpoint to receive the current position from the boats since it receives it in the form of MQTT messages, through Paho MQTT library integration, by subscribing to the *CAM* message topic. This API is deployed in a Docker Container and handles MQTT message communication directly with another Docker Container running NAP-Vanetza, acting as a message broker between the tracking platform and the boats which are being tracked.

The main entity in our architecture, the Boats, also communicates directly with a Docker Container running NAP-Vanetza, although the former was by necessity (due to

problems configuring it as a bridge for all the boat's brokers), the latter is by design, as to simulate an embedded broker on each boat. As for the boats themselves, each is running the same script parallel to the other nodes in the network, inside their respective Docker Container.

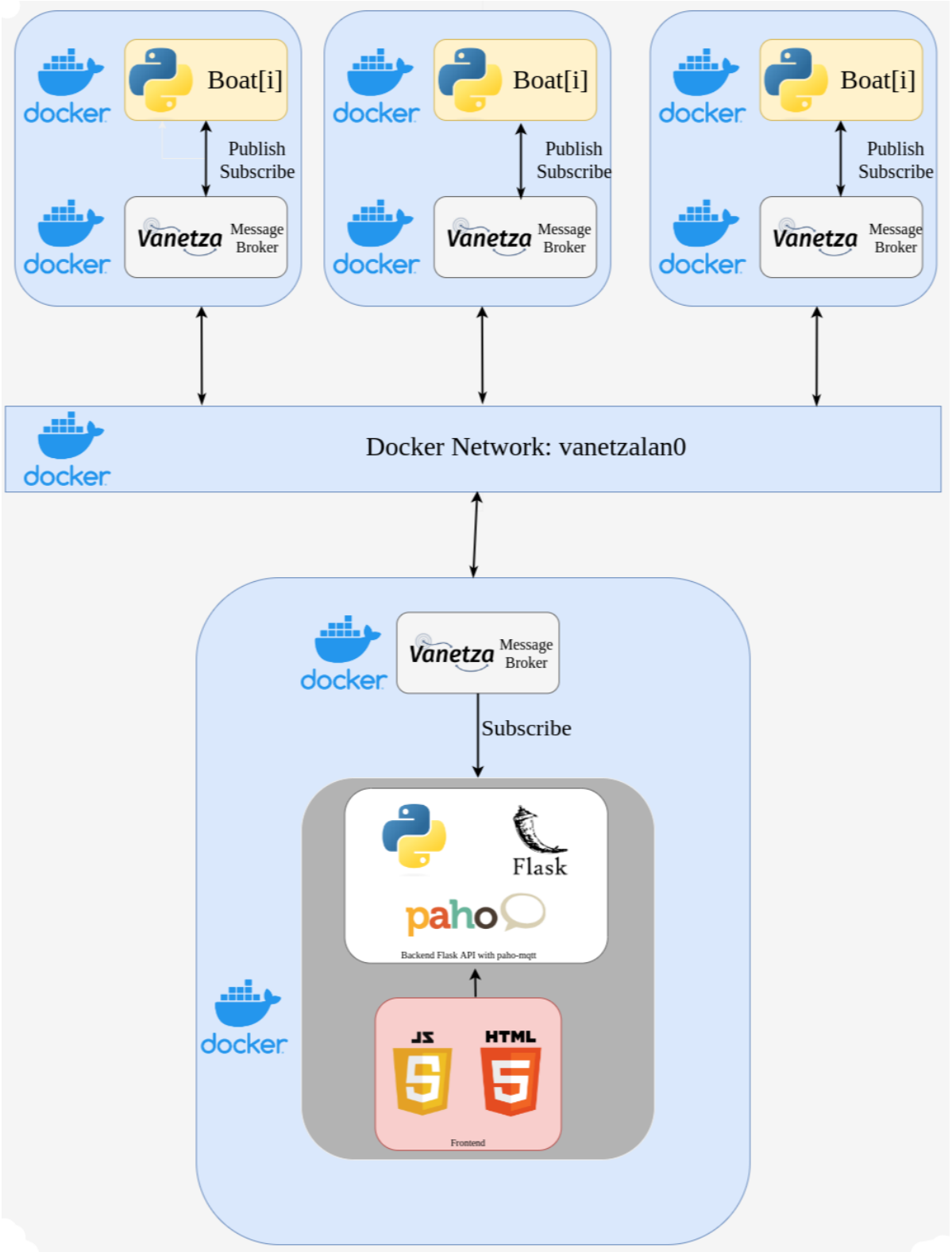


figure 1 - Architecture Diagram

Messages

As stated previously, communication between nodes is ensured through Message Queuing Telemetry Transport on 2 different topics. Each topic follows their specific ETSI C-ITS format: *Cooperative Awareness Message* structure for messages between the Tracking System and each Boat; *Decentralized Environmental Notification Message* for messages between Boats. Their logic is as follows:

- Cooperative Awareness Messages (CAM):
    - The Tracking System subscribes to these messages to show in real-time the location of the boat that is sending them, aggregates current coordinates to previously received and displays the course it has plotted;
    - On processing of each of the boat's simulated moves, the script publishes a CAM on this topic, which is received by the Flask API's broker to represent the new coordinate.
- Decentralized Environmental Notification Message (DENM):
    - All boats both subscribe and publish on these message's topic, making this topic work as a broadcast;
    - After a boat discovers a reference point of the maritime area, these messages are published on their topic and thus shared between boats, in order to notify the most recent discoveries.

It should be noted that not all mandatory fields in both CAM and DENM formats were used, leaving them to a default value that isn't verified neither by the sender nor the receiver, only the structure is verified by the Vanetza brokers. A simple communication diagram between 3 boats and the Tracking System (Flask API), is as shown in figure 2.

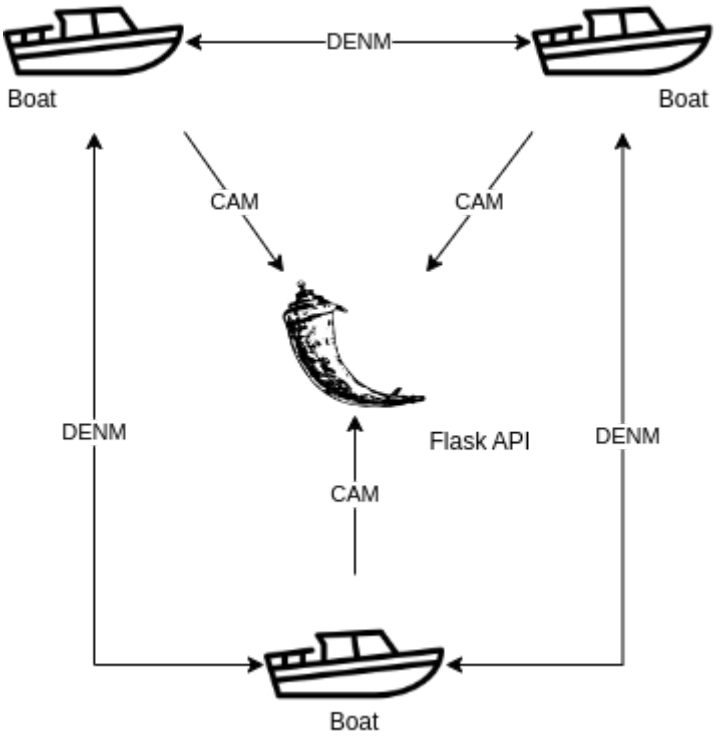


figure 2 - Communication Diagram

Boat Nodes

On initialization, each boat runs the same script deployed to its Docker Container, loaded with specific configurations passed by environment variables through Docker Compose. After that, the boat enters an infinite loop which will be henceforth referred to as its life cycle. In order to provide a clearer understanding of our implementation, the various stages of the boat's life cycle have been organized into distinct sub sections.

Environment Configuration

In order to ensure that the simulation can withstand multiple scenarios, we can dynamically define certain parameters through Dockerfile and Docker Compose.

Since this is something that must be the same for all boats, in the Dockerfile we can set the coordinates for the map area, which must be a list of 4 tuples passed as a string, and we can also set the map precision, which is the number of columns/rows our reference grid will have (see figure 3 for clarification), the more the precision value, the more reference points the grid will have and, in the simulation, this also translates inversely proportionally to our boat's vision range. We recommend 10 points per kilometer, being 50 meters the approximate range of our boat's vision.

As for the environment variables specific to each individual boat, we specify them in the Docker Compose file and they include the boat ID, the boat IP address, the boat's broker's IP address and the boat starting point coordinate, which are all self explanatory. As for the last one, the movement amount, it specifies how many coordinates the script will generate for every 100 meters to simulate movement from one point to another, for example, given a movement amount of 5, if the script calculates a distance of 100m it generates 5 coordinates, but if the distance is 140m, it will generate 7 coordinates. Since each move takes about 300ms, the more the movement amount, the slower the boat will appear in the simulation. Since we want speed in our simulation, we recommend 1 point per 20 meters, which translates to about 130 knots.

Before the boats can enter their main loop, a little more set up is needed, firstly the reference points as seen in figure 3 are calculated, with the specified precision, then the connection to the boat's broker is established, the DENM topic subscribed and the initial configuration ends with a publish of the first location to the Tracking System.

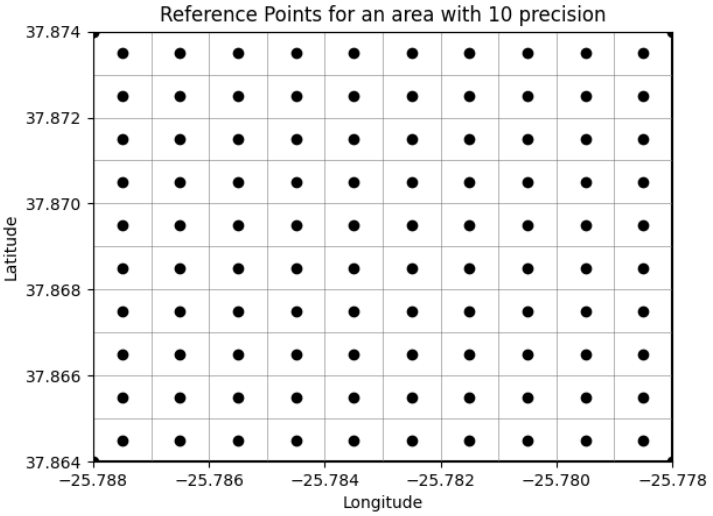


figure 3 - Reference Points for an area with 10 precision

Boat Life Cycle

Haven't yet made contact with any of the other boats, they start by calculating the closest reference point in the list of undiscovered points, then the movement amount is calculated from the current position to the elected point, generating then the appropriate number of coordinates for simulating movement. These coordinates are simply a list of increments from start to finish, by a given amount, which are then iterated and published one by one in a CAM for the Tracking System to receive, with an interval of 300ms to simulate the time it takes for the boat to move between them. During these actions, if the boat has received a DENM with the same coordinates as its current destination or a DENM that indicates all reference points have been discovered, then a flag is activated, ensuring the boat's movement stops. After relaying all positions the boat has been on, it checks if it arrived at the predicted reference point, if so, then a DENM message is built and sent to all other boats, indicating the discovery of this reference point, which can now be removed from the list of undiscovered points. Making this reference point, or the last published position (in case of interruption), as the boat's new position, the cycle starts once again, but in the case of there being no more undiscovered reference points, the cycle reaches its end condition. Before shutting down, the boat sends its starting coordinate once again, to ensure boats that booted a little bit too late get a chance to hear it.

Here we can see a simplified adaptation of the boat's algorithm:

Algorithm Boat Patrol
Input: Boat ID, Broker IP, Zone Corners, Map Precision, Movement Amount, Starting Position
Output: Boat Movement Coordinates

```
Initialization: Generation of Reference Points List, Connection to Broker
While(Reference Points):
    Calculate Closest Point
    Calculate Amount of Moves to Closest Point
    Generate Coordinates to Closest Point
    Publish Moves (CAMs)
    If (ForeignDiscovery && (Not RefPoints || ForeignDiscovery is Closest)):
        Stop Publishing Moves
        End If
        If (Arrived at Predicted Reference Point):
            Publish Discovery (DENM)
        End If
        Current Position = Last Published Move
    End While
    MQTT Thread Join
```

Persistence and Fault Tolerance

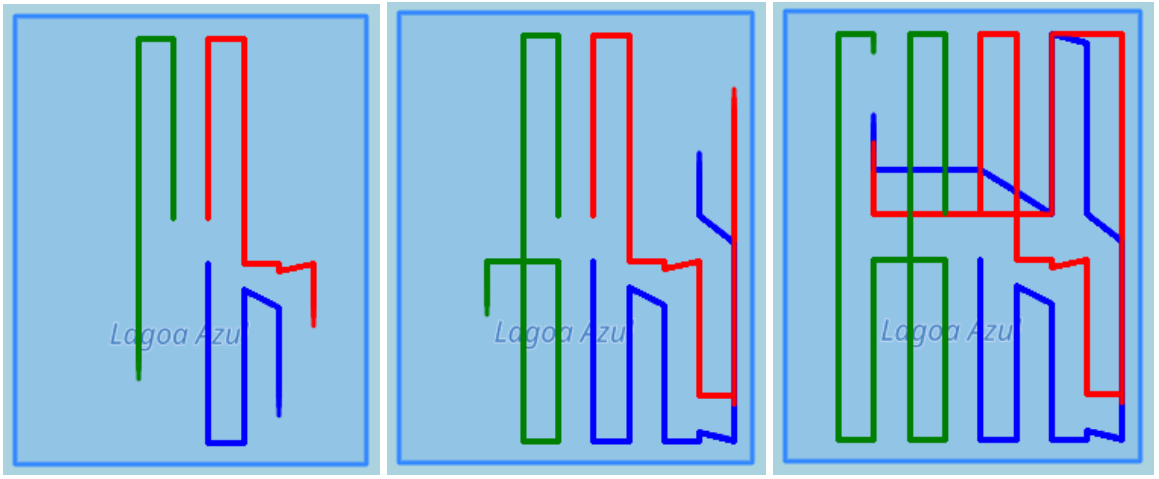
Should a boat suddenly shut down, it will remain in place, waiting for a pick up from the maintenance team and for the remaining boats to finish the job. We are assuming a full system shutdown, so we assume we can't command the propellers to sail to a rendezvous point, the maximum we can do is make sure the other boats don't sail through it in their patrol.

The system might be re-bootable over-the-air, and with persistence of data, we could even recover the boat's last known undiscovered reference points, but we would have no way of making sure this boat received the DENMs sent in the time it was offline, therefore, no way of making sure the boat stayed synchronized with the rest of the network. In summary, the exclusion of persistence was by design, to make the system more cost-effective, more performance and energy efficient and simpler.

Demonstration

A link with a video of a live demonstration of 3 boats, each starting on a third of the maritime area can be found here. The video is very self explanatory, each line is the course a boat is plotting, the boats are heading to the nearest undiscovered point, while blue plots a very well defined course, red decides to explore the bottom of the area, while green curves his way to the last remaining undiscovered points. Since the boats' speeds are all the same, green can't catch up to blue and ends up shutting down since all points have been discovered. Blue arrives at the last undiscovered point first and shuts down, since red is very close and had the same destination, upon blue's most recent discovery, red also shuts down, ending the simulation with the map fully discovered and close to no points visited twice.

These figures 4-6 are screenshots from another run of the simulation, but with starting points in the center of the area. We can see green go up, then loop back around discovering the left of the map. Meanwhile, red and blue almost converge, but calculate different closest points which makes them explore different areas but almost converging once again, which prompts blue to explore another area in order to avoid collisions since red is just a few meters ahead. After that, for a brief moment, red and blue have the same closest undiscovered point, but split once again. Ultimately, all three boats converge to the last unexplored point



figures 4, 5 and 6 - Screenshots from a run of the Simulation

Conclusion

This has been our implementation of an Ad Hoc network with nodes all working for the same goal, with fault tolerance, communication with messages that follow the ETSI C-ITS protocol for ITS-G5 channels in VANETs, with a fully configurable, scalable and dynamic environment used as test bed for a simulation of a real world application: automated boat patrols in hard to reach and dangerous places. All of the developed code, presentations and reports can be found at https://github.com/FastMiguel099/RSA-Project/.