University of Aveiro

Drone Network for Data Gathering

Autnomous Systems and Networks

Álvaro Frazão - 97912, Bóris Teixeira - 98472

Contents

List of Figures

Chapter 1
Introduction

The goal of this report is to explain and describe the steps taken during the development of the project for this course. The code developed for this project can be found at `https://github.com/alvarofrazao/rsa_dn` and the demo can be viewed using the following link: `https://youtu.be/H39ru4wpXwk`

1.1 Objectives

The main goal of the project was to develop a drone network that would allow for data collection in a certain geographic region using sensors and other gathering tools (digital cameras, IR sensors, thermal imaging). This network is meant to function in a way that is independent of the tools used to gather data. This would allow the network to be deployed in a variety of different scenarios without requiring too many changes.

1.2 Network entities

The network is made up of two types of entities:

- **Drone Unit** - Unit responsible for collecting data. Moves to targets defined and delivered to the Drone by the Core Unit.
- **Core Unit** - Center point of the network. It is the unit responsible for generating the target positions to which the Drone Units move to. It is also the unit which receives all data collected by the network

1.3 Network operation

The way in which the network operates can be describe as such:

1. The Core Unit generates the four cell grids for a given maximum distance to its position and calculates each cell center;
2. Core Unit sends a cell center to each drone;
3. Drone Unit calculates the path to its target and moves to it;
4. Drone Unit remains in the cell for a pre-determined amount of time, all the while sending activity data back to the core;
5. Drone Unit returns to the core;

Process repeats from 2 until all cells have been scanned.

1.4 Messages

In order to organize communication between several elements of the network, two types of messages were defined (both following the JSON format):

- ***CMs*** (Core Messages) - Message type sent by the Core Unit, used only to transmit the target positions to the drone units and therefore only contain coordinates.
- ***DMs*** (Drone Messages)- Message type sent by the Drone Units, used to update the Core's information about the sending Drone Unit, meaning it contains not only the Drone's position, but also its current state. Contains also a field for an IPFS CID that is used when the unit detects activity.

1.5 Technology

To develop and implement the network, we used *MQTT* brokers for unit to unit communication.

We used *Python* for the programming language in order to create the *MQTT* clients for each unit, and for the cell generation and path-finding algorithms.

We implemented an *IPFS* network for the transferring of files between the Drones and the Core.

We implemented the visualization software using a *Flask* web server running the *Leaflet* library.

Lastly, for the simulation, everything was implemented and interconnected in a *Docker* container environment.

1.6 Development steps

For the development of this project, we first started out by figuring out the communication infrastructure between each unit (in this case, both through *MQTT* and *IPFS*). Next, we decided on how to best approach the problem of distributing the drones geographically hence the use of the cell grid approach for the drone targets. Lastly, we implemented the *IPFS* network, wrote a script to simulate possible activity and developed the visualization website.

Chapter 2
System

2.1 Architecture

Considering everything that has been mentioned in the previous sections, the architecture of the project is represented in the following diagram:

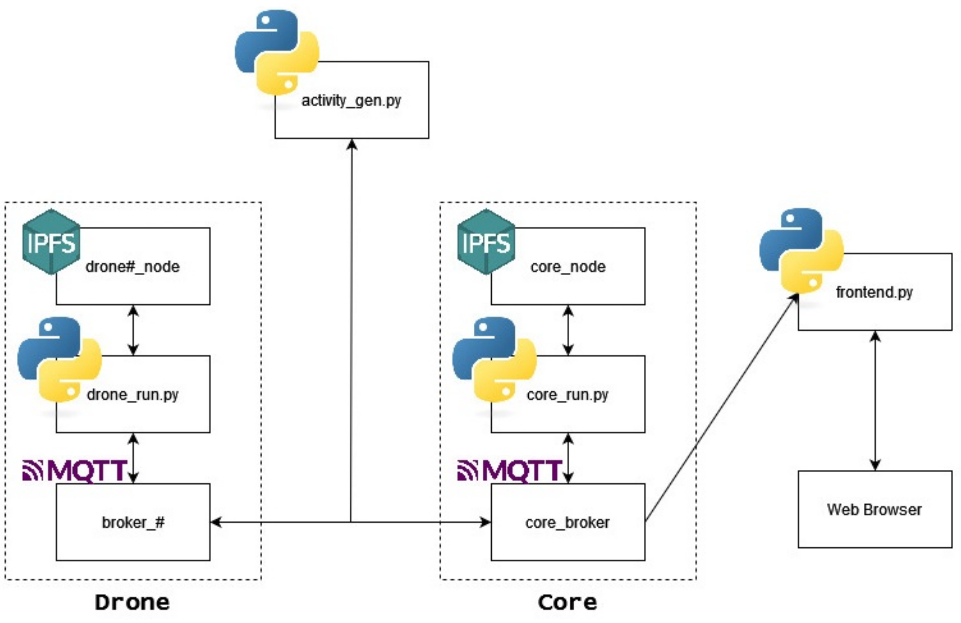

Figure 2.1: Architectural diagram

In order to preserve space, only the connection between the core and a single drone have been represented.

Taking into account what is presented in Figure 2.1, we can divide the project into two different layers:

- The message processing layer, to which the *Python* scripts belong;
- The communication infrastructure, made up of the individual brokers that each Drone entity has, alongside the Core unit's broker.

2.2 Communication Infrastructure

The communication infrastructure was envisioned in a way that gave each Drone unit their own individual broker and two clients connected to it: the drone itself, and the core unit. This approach allows for easier understanding of the message flow through the network and also allows for a more well defined message distribution approach as in we don't have to consider scenarios where drones receive messages that were not meant for them. This is of course an ideal scenario that does not consider the possible network issues such as message loss or connection loss. The brokers are organized as such:

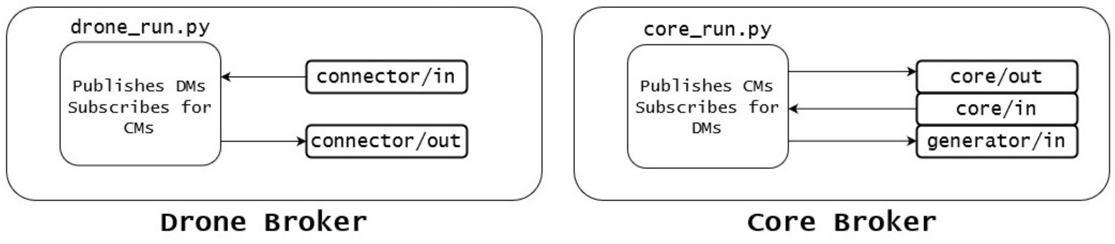

Figure 2.2: Broker diagram

Again, for the sake of space, only the individual broker designs were represented. In the case of our project, we deployed the brokers in a virtual *Docker* network (*vanet-zalan0*, 192.168.98.0/24) with the details of each broker being defined in a *docker-compose.yml* file. Each broker was given a unique ID and IP address according to the following list:

- core_broker - 192.168.98.20
- d10_broker - 192.168.98.10
- d11_broker - 192.168.98.11
- d12_broker - 192.168.98.12
- d13_broker - 192.168.98.13
- d14_broker - 192.168.98.14

As for the *MQTT* clients, they are instantiated on the *Python* scripts running on each unit, with the scripts having a client for each broker they need to connect to.

2.3 IPFS Network

In this project, a small scale *IPFS* network was implemented for the purpose of exchanging files between the Drone units and the Core unit. As mentioned before, the idea is for data gathering and aggregation to be made possible regardless of the sensor arrays the drones might possess.

It was then theorized that, for that effect, whichever sensors the units might have, their data would be kept as files that would then be transferred to the Core unit when the conditions were met: in the case of this project, an image file is requested by the Core every time a Drone unit detects any sort of activity.

2.4 Behaviour Scripts

Each unit has an associated *Python* script to control its behaviour (movement and message handling):

- ***core_run***: This script contains all the functions related not only to the Core Unit's message handling, but also the cell generation algorithm. It starts out by generating the target cell grid, then creates the *MQTT* client threads and finally initiates the Drone Unit movement. It continuously prints the current state and position of each Drone Unit and also periodically updates the *webviewer* application with the most recent drone positions;

- ***drone_run***: This script, like the previous one, contains all the functions related to the unit's movement, message handling and activity detection. Once it receives the initialization and first target messages from the Core unit, it starts its movement by calculating it's path as a series of N positions between its current position and its target. In the current implementation of our project, N corresponds to fifteen movement steps;

- ***gen_run***: Its only function is, for a given cell grid center and quadrant corner provided by the Core unit, to generate a random pair of latitude and longitude values transmits it to each running *drone_run* script. This serves as a way to emulate the possible activity that sensors present on a real drone would detect in a given geographical area.

2.5 Script functions

In the following listings, a functional description of the relevant functions present in each script will be presented.

2.5.1 *core_run.py*

- generate_cells: Generates the coordinates of each point relevant to the cell grid;
- gen_centers: Computes the center point of each one of the cells for a given quadrant;
- drone_connect: Prints result of drone broker connection attempt (successful or not) and subscribes to the appropriate broker queue;
- gen_con: Prints results on local broker connection (successful or not);
- drone_onmsg: Handles the reception of messages sent by the Drone units;
- gen_onmsg: Handles the reception of messages sent by the generator unit;
- initiate: Sends the first message to the drone units, containing the Core's position and
- send_coords: Sends message containing a Drone Unit's target position;
- init_gen: Sends message to the generator containing the position of the Cores and the farthest corner of the current quadrant;
- update_markers: Updates the drone positions on the *webviewer* application.

2.5.2 *drone_run.py*

- move: Generates the drones path for a given number of steps;
- receive_coords: Handles the processing of messages sent by the Core to check if they're target messages or initialization messages;
- detect: Checks if the position received from the generator is within its detection range or not
- on_message: Handles the reception of messages from the generator unit, calls the detect method, and if it returns true, sends a message to the Core unit containing the CID of the image currently present in the container for the Core Node to request from the *IPFS* network;
- update: Sends message to the Core containing its current position;
- on_connect: Prints result of the attempt to connect to the local broker;

2.5.3 *gen_run.py*

- gen_con: Prints result of the attempt to connect to the drone unit broker;
- gen_msghndl: Handles the reception of messages from the Core unit. Responsible for changing current number generation boundaries.

2.6 Web Viewer

The web visualization software for this project consists of a simple *Flask* application that displays the most recent position of the Drone units recorded in memory. These positions are passed to the application by the *core_run* script through the Core Units broker, every time it receives a position update message from one of the drones. The display was made using the *Leaflet* library and some additional *Javascript* commands to allow for automatic update of the web page as opposed to using manual refreshing.

Chapter 3
Simulation

For this implementation, it is important to note that the overarching behaviour of all the entities in the network, is not random. The only existing randomness in this simulation is the number of detections made by each drone unit during their life cycle. This means the results of the simulation will always be mostly similar across all runs, with the differences between runs not being anything relevant to the analysis of its results.

3.1 Simulation Flow

The simulation is initialized when all the containers that run the *Python* scripts are started. From this point on, the Core unit generates the cell grid, the center of each cell and performs the initialization of each Drone unit by passing the starting message with their first position (position of the Core unit). It is only at this point that we can consider that the simulation has truly begun. After sending the initialization message, the Core unit then sends to each Drone a target position. Upon receiving this target, the drones will compute the path needed to take from their current position to their target and move to it. Once they have arrived, they will hold their position for a previously a random amount of time between 7.5 and 15 seconds. Once the waiting period finishes, they start moving back to their starting position and once they arrive, they will wait until they are given a new target position by the Core. This process repeats until all the cells in all 4 quadrants have been visited, at which point the core will no longer assign targets to any unit. It is important to note, that during the simulation, each drone is continuously updating the Core unit with their current position and state.

3.2 Results

The key takeaways from the execution of the simulation can be done, in a qualitative manner, both from the drone movement performance point of view, and the activity detection performance point of view :

- Detection performance - Due to the fact that the activity that is supposedly detected is random, we can conclude that the amount of times a drone detects something is relative to each run which means that making a concrete performance assessment is inconsequential because the detections of each drone can vary greatly from run to run.

- Movement performance - Considering that target assignment is done in a sequential manner, the first drone will always visit more cells than the other four. This because, in each set of 5, generally speaking, the first one of the set is always the closest one to the core. With this in mind, we can conclude that there is a unit that is ”working” more than the other four, which points to an uneven workload distribution in regards to cell assignment.

Chapter 4
Conclusion

4.1 Conclusions

Considering not only the project as a whole but also its development process, there are several things that are worth pointing out. In terms of the development process, the attempt to make the network general and usable for different kinds of sensor arrays mounted on the drone, made it difficult to assess the networks performance because it lead to a lack of clarity. This was because "activity" and the means to detect it, where not concretely defined. This abstraction from the data gathering methods, also lead us to have a very simplistic function for the Drone units, which in a real world scenario, would have a lot more things to interconnect such as unit to unit communication, data gathering and processing and the movement of the unit itself. In a virtual environment, considering what was previously stated, a lot of these things are hard to truly visualize and, as a consequence, figure out how to implement. In terms of the project itself, it is to note that the network requires a constant connectivity between each unit and the core, which in a non virtual environment, would require not only a routing protocol between each drone, but also a target assignment that would have to take into account the maintenance of this network.

4.2 Demonstration

Alongside the file of this report, there is a video file containing a recording of a run of the simulation. On the left is the terminal window of the core Unit, where the positions, number of detections and unit states are all printed to and on the right is the web visualization window where each drone is represented by a marker. Due to the inconsistency between Drone unit message rate, and the "sampling" rate of the webviewer, unexpected marker jumps are expected. The demo runs until all Drone units have registered five complete runs.