Good practice Guide for developing Agent-based Sensor networks

Bang Xiang Yong

University of Cambridge

1. Introduction

Sensor boards could be distributed geographically, and each could host a number of sensors measuring different physical quantities from temperature, pressure, electric current, water level, to air pollutants. To this end, agent-based systems provide a convenient software framework for managing distributed and heterogeneous sensor streams and their analysis. The Agent-Oriented Programming is a distinct extension to the Object-Oriented Programming paradigm, specifically embedding the methods and infrastructure to (1) handle sequential data, (2) route messages between software agents, and (3) grouping of agents.

This document provides a guideline in developing agent-based systems using the agentMET4FOF Python package for sensor networks. An example output of the package is shown in Figure 1. The datasets and agent classes described here are accompanied by easy-to-follow supplementary open-sourced code in a Github repository. Extensions could be made to incorporate metrological methods to ensure traceability and quantify uncertainty.

Readers are expected to have a basic understanding of sensor networks, software engineering and Python programming.

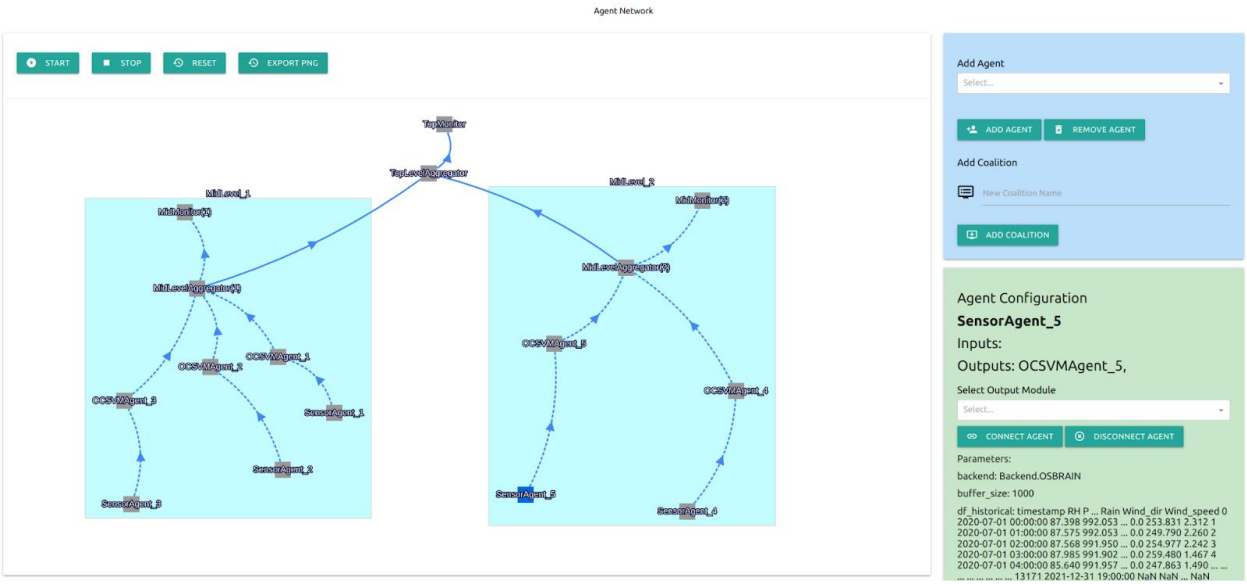

Figure 1: Dashboard illustration of the connection of agents in the sensor network topology. Each gray box represents an active software agent. The flow of data through the analytical

pipeline is shown by the direction of the arrows. Agents in the hierarchy are further grouped into coalition as depicted by the enclosed blue box in the agent network.

2. General workflow of modelling agent-based sensor network

The general workflow in modelling sensor networks as an agent-based system is depicted in Figure 2. In practice, the steps involved may not be strictly sequential and iterations are expected. One benefit of agent-based modelling is its reconfigurability and reusability, allowing developers to add and modify the agent's behaviour and connections, while ensuring the designed agents could be used again for other sensor networks. We describe the workflow as follows:

Figure 2. General workflow of modelling agent-based sensor networks

I. **Specify goal of monitoring.** Developers need to consider the stakeholders' requirements in the context of the use cases.

II. **Identify number of sensors and their locations.** It is recommended to model each sensor board such as a Raspberry Pi as an agent, which can be mounted with multiple sensors.

III. **Specify analytical methods.** The types of statistical algorithms to be deployed can be categorised into supervised or unsupervised learning. In supervised learning, a target label is needed for the model to be trained on to make a prediction. Such labels may not be easy nor cheap to obtain. Alternatively, when no target labels are available, one could opt for unsupervised learning which is widely applicable to sensor networks data where no ground truth data is available.

Moreover, we can further specify whether the model is trained offline or online. In offline mode, the model is trained and saved on a remote PC and then transferred to

the agent to be loaded. On the other hand, online training allows the model to be updated on the fly as new data pours in. One can view a classical Z-score model as an online unsupervised method which computes the Z-score of raw sensor measurements on a rolling window basis. Thereafter, a threshold could be applied to flag the processed values as anomalies.

IV. **Determine ways to aggregate processed or raw sensor data.** More often than not, aggregation is required to combine the readings of sensors which are distributed geographically, and as such, could be arranged in a hierarchical manner. For example, in the use case of solar power generation, the sensors could be monitoring the power of individual generators, which are located in a farm of such generators. Moreover, we can further group these farms of generators by their locations.

    Aggregation could also be needed to "compress" the processed data from homogeneous sensors to reduce the network load.

V. **Specify behaviour and connection of agents.** Once most of the details of the sensor network topology and their analytical methods are specified, we can model the behaviour of agents and their connections to reflect the intended data flow and execution of analytical algorithms.

3. Use cases

While the implementation of agent networks is generalisable to most use cases, this document considers five specific examples: monitoring of

(1) air quality

(2) environment

(3) energy use

(4) solar power generation

(5) water resource distribution.

The datasets for (2), (3), (4) and (5) are open-sourced real-world data and could be downloaded from kaggle.com. More details are available in the provided links in the Appendix.

4. Design of Agents and methods

The agent-based system comprises of a hosted nameserver with a network IP address and agent classes which are instantiated as software processes running concurrently. These agents could run in distributed operating systems as clients to the nameserver and communicate via a common protocol, ZeroMQ.

Developers would have to design their own agent classes specific to each use case. To further ease the development, common classes are designed to encapsulate the information flow of sensor networks from streaming, processing, aggregation to display. As a note, these agent classes inherit the base agent class `AgentMET4FOF` provided by the agentMET4FOF package which provides common methods for communicating among agents and updating on every time tick.

Generally, the agents have inherited methods which could be overriden by the developers. Firstly, an initialisation method `init_parameters` is called once upon launching. Then, the method `update_step` is repeated on every time tick specified by the user (defaults to 1 second). Asynchronously, when a message is passed, the receiving agent will call the method `on_received_message` which has the message as the method's argument.

Next, we specify the designs of agent classes common for sensor networks: SensorAgent, AnalyticsAgent, AggregatorAgent, MonitorAgent, depicted in Figure 3. These agent classes serve as templates and the specific details would have to be customised to each use case.

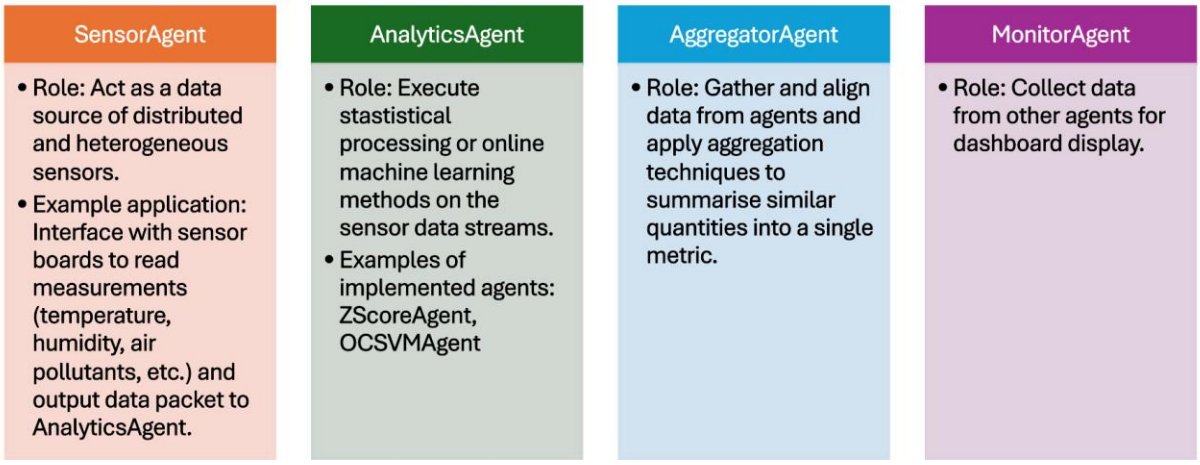

Figure 3: Common agent classes for modelling sensor networks

SensorAgent

The main task of the SensorAgent is to read sensor measurements from a data stream and output to an AnalyticsAgent. Importantly, the SensorAgent can read multiple types of quantities and is not constrained to only one type; that is, the SensorAgent is capable of representing a sensor board with heterogeneous sensors (e.g. temperature, humidity, etc.). In a simulated setting, we load the sensor measurements from .csv files as data frames and then read them row by row, assuming the data have been sorted by their timestamps and arrive sequentially. In deployment, the user will need to provide a method to read from a socket or datastream.

AnalyticsAgent

The AnalyticsAgent awaits messages from a SensorAgent to process the raw measurement values in a sequential and incremental manner. While most analytical models are often implemented in an offline, batched manner, it is essential for a sequential, online version to be provided for it to be deployed online with the agent framework. Specifically, this means a method which accepts the arrival of new measurements needs to be provided.

To facilitate the transition from offline data modelling to deployment, we recommend the best practice is to implement a model class capable of processing in batch and sequential manner. Such a class could be decoupled from the AnalyticsAgent for better manageability. For instance, a `process_sequential` and a `process_batch`method could be implemented which accept an array of values. Then, test scripts should be developed to ensure these methods yield similar results by comparing the outputs. The `process_sequential` should have an `update_step`

which processes the entries in the array sequentially; this way, the agent class will be able to make use of the `update_step`.

Next, we address several common cases could arise with machine learning methods in the following points:

- For algorithms that require fitting on a large-scale training data, we suggest that the model could be trained and saved offline then loaded by the AnalyticsAgent upon instantiation.
- Alternatively, the AnalyticsAgent can store the streamed data received from the SensorAgent in a buffer until a desired amount of training samples has been attained. Then the model is trained using this buffered data.
- If the model supports online or incremental learning, the AnalyticsAgent could update the model with every mini batch of measurements stored in a buffer.

The use of a buffer is ubiquitous to manage the storage of measurement values before applying statistical methods. Moreover, it is also common to expect that the AnalyticsAgent could be deployed in a system with limited memory storage. In this regard, the buffer has a limit to the number of measurements it can store at any time. In addition, the buffer has to append and eliminate entries in a First-In-First-Out (FIFO) manner, while keeping track of the heterogeneous quantities. These requirements are conveniently covered by the use of the `AgentBuffer`.

While the agent class described may not be able to house the use of all variants of machine learning algorithms, the provided scripts demonstrate examples of AnalyticsAgents for computing Z-scores and an online machine learning method, One-Class Support Vector Machine.

AggregatorAgent

After processing the raw measurements, one common task is to aggregate these processed values to provide a holistic view of the distributed sensors. The role of the AggregatorAgent is to accept the outputs from multiple AnalyticAgents and compute an aggregate statistics. For starters, one form of aggregation would be computing the average of the processed values, reflecting the mean of the distribution of processed values. More sophisticated strategies could also be considered to reconcile these values. In a sensor network with a hierarchical nature, AggregatorAgents from a lower level could be connected to an AggregatorAgent of a higher level in the hierarchy.

Coalition of sensor agents

SensorAgents, AnalyticAgents, and AggregatorAgents could be grouped by their geographical location or nature of tasks to improve manageability and to reflect their connection in a hierarchical topology. To facilitate this, we make use of the Coalition class which is a collection of agents and provides easy visualisation on the dashboard.

5. Conclusion

In this document, we have described the general design of agent-based system using agentMET4FOF package for a set of common sensor network setup. The general procedures towards modelling the sensor network at hand are described, along with the design of agents. Specifically, the issues of incorporating online machine learning algorithms, connecting agent with aggregation, and behavior of agents are described. Further technical advancements could be made to incorporate metrological design to ensure traceability, explainability, and quantify uncertainty. To gain better clarity, the accompanying code base should be studied and executed.

Appendix A

Links to datasets

1. Solar power generation: https://www.kaggle.com/datasets/anikannal/solar-power-generation-data
2. Environmental monitoring: https://www.kaggle.com/datasets/garystafford/environmental-sensor-data-132k
3. Energy use monitoring: https://www.kaggle.com/datasets/loveall/appliances-energy-prediction/data
4. Water resource monitoring: https://www.kaggle.com/datasets/edomingo/catalonia-water-resource-daily-monitoring