Real-Time Monitoring and Active Control of Autonomous Agricultural Robot Trajectories Using an Edge-Fog Architecture

Mohammad Kassir[1,3], Sandro Bimonte[1,3(✉)], Robert Wrembel[1,3], Mohamed El-Ouati[1,3], and Mahmoud Sakr[2,3]

[1] INRAE, Clermont-Ferrand, France
{sandro.bimonte,mohamed.el-ouati}@inrae.fr,
robert.wrembel@cs.put.poznan.pl
[2] Poznan University of Technology, Poznan, Poland
mahmoud.sakr@ulb.be
[3] Université libre de Bruxelles, Brussels, Belgium

Abstract. Smart farming concerns the usage of autonomous robots and sensors in the field to implement agro-ecology practices. Agriculture stake-holders are supported by supervision and control systems that allow for monitoring real-time data by means of Data Stream Management Systems (DSMSs), and remotely control robots in the field. However, processing in real-time streaming trajectory data that come from autonomous robots presents significant challenges due to the large volume of data generated by robots, and their bad quality since communication networks deployed in rural area can present several problems (e.g., instability and congestion). Furthermore, existing lightweight DSMSs do not support spatial data, and the proposed supervision systems are based on cloud-fog architectures, which do not provide effective solutions for trajectory data stream analysis within bad quality communication network. Therefore, in this paper, we extend [14], by proposing a hybrid edge-fog architecture and computation framework for robotic trajectory data analysis, which combines distributed and non-distributed DSMSs to manage trajectory data more efficiently. In particular, we extend the lightweight DSMS *Esper* with spatial operators, and dynamic frequency mechanism, at the edge level to process complex queries over trajectory data stream of the robots. Our architecture uses *Geoflink* at the fog, i.e., in the farm. A distributed computation approach for complex queries has therefore been implemented to split queries over the edge and the fog. We validate our proposals using experiment in real-life conditions.

Keywords: Data stream management systems · agricultural robots · anomaly detection · continuous queries · real-time data processing · edge processing

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2026
O. Gervasi et al. (Eds.): ICCSA 2025 Workshops, LNCS 15890, pp. 263–280, 2026.
https://doi.org/10.1007/978-3-031-97606-3_18

1 Introduction

Agricultural robots are autonomous machines designed for farm works. They are equipped with sensors to perceive their environment and actuators to carry out tasks, in order to operate independently of other robots and requiring no direct human intervention. These robots typically consist of a mobile base with a navigation system and a set of agricultural tools that can be mounted directly, semi-mounted, or towed. These robots are more and more frequently used in agro-ecology. Agro-ecology emphasizes environmentally sustainable farming practices. To support this, farmers and stakeholders need a robust system for supervising and remotely control agricultural robots [14]. Such a system should allow remote monitoring of robot progress in the field, as well as ingesting and analyzing data from various field devices, that can be classified as:

- *sensor data*, which are delivered in real-time from environmental sensors, e.g., temperature, wind speed, soil humidity;
- *odometry data*, which represent robot's position, movement, and other mechanical parameters; such data are continuously updated, reflecting current positions of robots in fields;
- *contextual data*, including spatial information (e.g., agro-field boundaries), and other farmer-specific details (e.g., the type of crops being harvested).

Sensors and odometry data are continuously generated as data streams and they are accessed in real-time, requiring a system that can efficiently handle these dynamic streams. For instance, it is crucial to track a robot's movement (speed, GPS location) and monitor weather conditions (temperature, humidity), since farmers should be able to control the right execution of agricultural tasks, monitor the robots performance and avoid any security human and animal risks. Moreover, such supervision systems must also provide remotely control capabilities to allow farmers to action robots from their working position to change the robots behavior when it deviates from the planed trajectory (e.g., robot has a delay), to guide it when robot is not able to autonomously continue its work (e.g., robot stops in presence of a obstacle), etc.

Data Stream Management Systems (DSMSs) are used in various application domains, such as traffic control, health, etc. to process scalable streaming data in real-time. DSMSs process data streams by means of continuous queries. A continuous query runs repeatedly at a predefined frequency. It incrementally processes results on data collected within a specified time window. For instance, a query might be configured to calculate the minimum speed of each robot every 5 s (i.e., frequency), based on data collected within the preceding 20 s (i.e., time window). Common aggregation operators, such as min, max, avg are used to summarize numerical values within a given time window.

However, handling agricultural robotic data with DSMSs raises some significant challenges, resulting from: (1) the high frequency of ingested data, (2) the need for real-time computation, and (3) poor quality (network instability and congestion) of a communication network deployed in agricultural fields, which can cause data loss. This loss deteriorates the integrity of the data, resulting

in inaccurate monitoring indicators, which is critical to maintaining operational safety and efficiency. Moreover, DSMSs come with a set of predefined queries, and they do not allow end-users to modify them, for example, changing the frequency of the query or the aggregation functions used.

To overcome these challenges, in this paper we extend the architecture proposed in [14] with a new **edge-fog** approach for monitoring of agricultural robotic data. Edge computing refers to processing data closest to the data source, e.g., on a device itself. It minimizes latency, improves real-time performance, and reduces bandwidth needs. Fog computing is a layer between an edge and cloud. It provides a balance of processing power and centralized management. In particular, we extend [14] with the computation of queries on streams of robot trajectory data in a distributed way between the edge and the fog. To this end, we extend the analysis capabilities of the *Esper* DSMS with trajectory types and operations. This is achieved viaintegrating it with the in-memory spatial DBMS *H2GIS*. We deploy it at each robot (i.e., edge level) to execute local queries. At the fog level, we use the Spatial DSMS *GeoFlink* to compute and aggregate data and Esper query results coming from the robots.

Moreover, we propose a dynamic time frequency mechanism that allows to change the frequency that robots send data to *GeoFlink*, in order adapt the monitoring system to the on-going situation of the robots. This dynamic time frequency mechanism can be controlled using a rich palette of predefined rules at the robot level that are automatically triggered. We also have implemented a mechanism allowing the end-users to remotely change the frequency of the continuous queries when need to support advanced control capabilities. To evaluate our proposal, a set of representative trajectory queries is proposed for robots monitoring in the experiments.

The paper is structured in the following way: Sect. 2 presents the motivation scenario and the definition of trajectory data and queries. Section 3 presents the architecture we extended, and the implementation of the continuous queries is shown in Sect. 4. Experiments are presented in Sect. 5.

2 Robots Trajectory Stream Queries for Monitoring Agricultural Autonomous Robots

In this section, we present the definition of a robot trajectory data and different kinds of continuous queries over such data for the supervision and control of agricultural autonomous robots.

2.1 Monitoring and Control of Agricultural Autonomous Robots

An agricultural robot is an unmanned ground vehicle equipped with sensors and actuators, capable of autonomously and safely performing one or multiple tasks in a farm field. It typically consists of a locomotion system integrated with a navigation module and an agricultural component, which may include mounted, semi-mounted, or towed implements. These robots can be designed for a specific



Fig. 1. Visualization of robots data. (a) The display show the robot pre-defined trajectory, and the robot real-time location. The depicted U-turn maneuver is one of the complex movements the robot can do, which might incur problems in implementing it. In (b) the end-user has a control device and a camera view to be able to control the robot trajectory when needed.

task, such as weeding or harvesting, or they can function as tool carriers for various equipment configurations for tasks like tillage and seeding. Monitoring and control systems must provide end-user support with a visual interface that is able to show in real-time different indicators, and to remotely control the robots in the field as shown in Fig. 1. In particular, it is necessary to monitor:

- *Robot malfunction and risks*: Agricultural robots operate in rural environments, often under harsh weather conditions and on challenging terrains (e.g., uneven or rocky ground), which may lead to technical failures. Moreover, robots can work in risks situations, such as the collision of robots, or with humans, etc.
- *Task alerts*: Robots are programmed to a particular predefined task in the field. Although robots are programmed to follow the predefined trajectories, they can autonomously adjust their paths to avoid obstacles like animals, tree branches, and rocks. These adjustments, however, may cause delays, or deviating from the predefined trajectory.

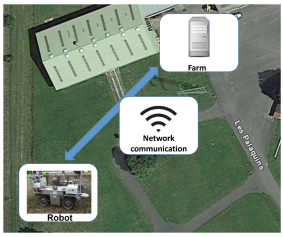

Fig. 2. Field configuration.

Typically, robots are deployed in the field, which is equipped with some communication networks as shown in Fig. 2. These communication networks can be

Internet, Wi-Fi ad-hoc network, etc. These networks are particularly challenging in farms since cellular network coverage may be not sufficient, or the bandwidth is low, which can generate data loss and by consequence a bad computation of monitoring indicators and impossibility to remotely control the robots. In the context of robots supervision, we have identified some relevant data-based indicators defined as a set of continuous queries:

- *Robot malfunction and risks*:
  - **Q1**: *Every 5 s, find robot Alpo's top speed in the last 5 s*;
  - **Q1a**: *Every 5 s, find top speed of all robots in the last 5 s*;
  - **Q2**: *Every 5 min, compute the total number of robots in a particular geometrical zone of the field in the last minute.*
  - **Q4** *Every 10 s, find possible collisions among all robots.*
- *Agricultural tasks' alerts*:
  - **Q3**: *Every 5 min, compute the delay of every robot according to their predefined trajectories in the last minute.*
  - **Q6**: *every minute, compute the traveled distance for every robot in 'Plot1' in the last 120 s.*

A visual interface for a supervision system must implement the *perception* level of the *situation awareness* (SA). Perception has been defined as *the level responsible for perceiving the status, attributes, and dynamics of relevant elements in the observed environment. The most elementary level of SA, involves the processes of sensing, monitoring and simple recognition, which lead to an awareness of multiple situational elements objects, people, systems, environmental factors and their current states and characteristics* [18]. The goal at this level is to maintain a constant and accurate understanding of the environment, which is foundational for higher levels of SA that involve comprehension and projection of future states.

In practical terms, for a supervision system overseeing robots in a field, the implementation of this SA can be achieved via continuous queries that report on the robots' status. For instance, a query (**Q1**) might check whether a robot has ceased movement for the past five seconds, potentially indicating a mechanical issue. Here, Continuous monitoring is key for real-time situational awareness.

However, the stream update frequency (*frequency parameter*) impacts user cognitive load. To address this, a **dynamic frequency** approach adjusts frequency based on result normality. For instance, if a robot's speed drops to zero (**Q1**), the system increases updates to every second, boosting situational awareness and enabling timely interventions.

To further optimize data flow and situational response, the concept of a backward channel from Fog to Edge in the typical edge-fog-cloud architecture is introduced. Unlike the traditional one-directional flow (from edge to fog, then fog to cloud), this two-way communication allows for adaptive data transmission based on the context, which could be governed by rules such as:

- *Spatial rules:* Certain zones, like near a field's boundary or a water body within it, may require more frequent updates to prevent the robots from straying off course or entering hazardous areas.

- *Event-based rules:* An increase in a robot's speed beyond a predefined threshold could trigger more frequent data transmissions.
- *Pattern-based rules:* Complex maneuvers, like a U-turn, which might increase the risk of incidents, could necessitate a higher data update rate to closely monitor and control the robot's actions.
- *Fog-control:* Fog, as farm controller, should be able to override default settings to adjust data flow as needed.

2.2 Robot Trajectory Point Data Type

In batch processing of trajectories—e.g., at the fog or cloud—a trajectory type maps time to space (e.g., in [15,21]). A robot trajectory is a time-ordered list of trajectory points. Figure 3 shows two example trajectories. In a streaming setting, trajectory points arrive incrementally, and the DSMS groups them into windows for processing. A trajectory point includes the following main attributes:

- *longitude* and *latitude* represented in the WSG84 coordinate system;
- *the set of numerical attributes*, which represent odometry data collected by the robot, as for example the *speed*;
- *timestamp* when data was collected by the robot, i.e., valid time.

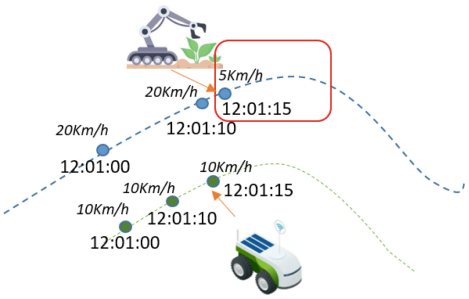

Fig. 3. Example of two robots trajectories, where one is represented in blue and the other one in green. The GPS position is represented by a colored point with the associated timestamp and their associated speed values (e.g., for robot blue: *12:01:00*-20 km/h at the first position; *12:01:10*-20 km/h at the second position; and *12:01:15*-5 km/h at the third position). (Color figure online)

2.3 Continuous Queries on Robots Trajectories Streams

A *continuous query* over a stream of robot trajectories involves the following:

- ***data sources*** - the specification of a table(s) (e.g., relational data describing an agro-field) and a stream(s) (e.g., a robot trajectory) to be queried, including means for joining them (if applicable);

- ***data selection*** - the definition of attributes to be extracted from the data sources (e.g., speed). In particular, *data selection* can imply different kinds of selection predicates, like: *alphanumeric* (e.g., robot name is "Alpo"), *spatial*, i.e., the usage of topological operators (e.g., the blue robot position is inside the red polygon, as depicted in Fig. 3), and *spatio-temporal* i.e., the usage of spatial and temporal operators (e.g., are robots blue and green at the same position at time *12:01:00*?).
- ***data aggregation*** - aggregation functions can be numerical or spatio-temporal. *Numerical aggregation* computes values like max, min, or avg using standard SQL functions or complex algorithms. *Spatio-temporal aggregation* focuses on the trajectory's points and timestamps.
- ***windowing*** - the definition of a time window (e.g., sliding, tumbling) and its parameters for evaluating a query (e.g., withing the last 10 sec);
- ***dynamic frequency*** - the definition of a frequency used to trigger the query (e.g., every 2 sec). The frequency can be defined using a set of rules evaluated on the trajectory data or can be remotely set up by the end-user.

An example of a continuous query using a numerical aggregation is: *every 5 s (frequency), compute the max speed of robot blue within 15 sec (window)*". Referring to the scenario shown in Fig. 3, at time *12:01:15*, the result is 20 km/h, since from *12:01:00* to *12:01:15* the max speed is 20 km/h at time *12:01:00* and at time *12:01:10*.

3 System Architecture

In this section, we present the main components of the *LambdAgrIoT* proposed in [14] (Sect. 3.1), and then we detail our extension for the effective edge-fog processing of robots' trajectories stream queries (Sect. 3.2).

3.1 LambdAgrIoT Overview

LambdAgrIoT has been proposed by [14] for collecting, integrating, and analyzing agricultural robotic data. It is a multi-layered architecture based on the *Lambda* architecture, and it is composed of the following main layers:

- **Data Producers Layer** includes a meteorological station and autonomous robots, feeding the system for real-time and historical analysis.
- **Speed Layer** processes real-time data streams produced by the *Data Producers* layer for decision-making. It is based on the DSMS *Apache Spark Streaming* coupled with *Apache Sedona*. *Apache Sedona* allows spatial data processing. The results of continuous queries are then visualized by means of **Real-Time Monitoring Dashboard**, which is a visual web interface.
- **Message Queuing Layer** serves as the backbone for data flow management within the architecture, ensuring efficient and reliable data transmission across different layers. It is based on *Apache Kafka*.

– **Batch Layer** supports long-term storage and analysis of historical data using *PostGIS* for spatial data management, and it implements a data warehouse for supporting OLAP queries.
– **Serving Layer** merges data from the *Batch* and *Speed* layers that feed an online scheduling algorithm implemented with CPLEX.

3.2 Extended Edge-Fog Layers

In this section, we detail the extension of *LambdAgrIoT* for computing continuous queries over trajectory data stream by means of an edge-fog approach. In particular, we have redefined the *Data producers* and *Speed* layers as shown in Fig. 4. The main idea is to compute the continuous query using the partial computation approach as described in [13], which consists in splitting the computation of the query in different parts at the edge level, and then aggregate them at the fog level. In our proposal, a continuous query can be computed: (i) totally at the robot level (i.e., edge), (ii) totally at the fog level (i.e., at the farm), or (iii) split accross the edge and fog levels.

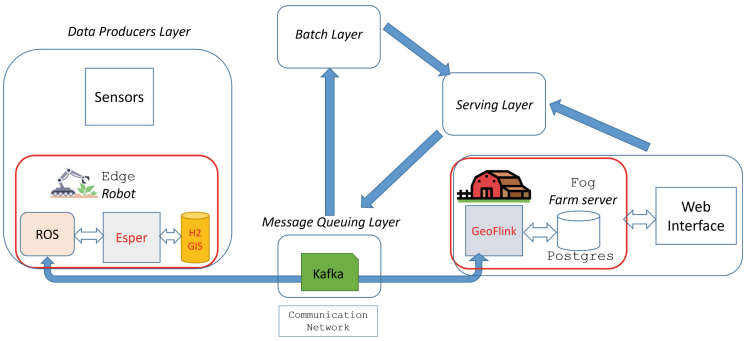

Fig. 4. Extended *LambdAgrIoT* architecture: in red rectangles the modified components of the architecture.

Data Producers Layer. A main contribution in this paper is that it extends [14] with the capabilities of Edge processing. In the extended architecture, edge devices are considered as processing units, in addition to their function as data producers. We therefore design a software stack for the edge devices, i.e., robots, in order to equipe them with query answering capabilities. Processing streaming query at the edge level needs for DSMS that are lightweight. Indeed, robots come with common laptop machine that have no-high performance physical devices such as RAM, CPU and disks. Moreover, the maintenance of the software of robots fleets is a tedious and time consuming task when the number of robots is high or this task is left to the farmers. Then, the usage of complex DSMS such as distributed one is not adapted for deploying the DSMS at the

robot level. This criteria plus the active maintenance of the DSMS software, and the need for the usage only of open-source software in our architecture lead to use *Esper*. A comparison of existing DSMS is provided in Sect. 6. From this analysis, *Esper* appears to be the most suitable choice because it is lightweight and can be deployed on robots. Furthermore, it has advanced pattern recognition capabilities, which are necessary for processing complex data (such as robots data), and it is actively maintained and updated.

However, Esper does not support spatial data processing. Therefore, we extend Esper by creating a new trajectory point data type and an associated set of methods. These methods implement spatial operators by means of the native functionalities offered by *H2GIS*. *H2GIS* is an in-memory database supporting the storage and querying of spatial data. It provides better performance for data access than classical disk databases [17], then it is compliant with real-time performance required by robots monitoring queries. Moreover, it is a lightweight system and by consequence its maintenance is also easier. Then, these spatial operators are used in the continuous queries methods offered by Esper to set frequency and window. Implementation details are provided in Sect. 4.

Further, our robots are equipped with ROS (Robotic Operating System). ROS provide a software framework for robots control and management. ROS uses a message subscribe protocol, which we use as a message queue that to handle messages from the fog. Other light message brokers systems could be also deployed. However, ROS allows to develop ad-hoc functionalities, it does not come with DSMSs functionalities. Thus, the usage of Esper plus HGIS allows to easy deploy continuous queries on robots without any additional ROS implementation. Moreover, this loosely coupled solution with Esper plus HGIS on the top of the robot software allow for exploiting also other robots that do not use ROS, which makes our proposal more generic and flexible.

Speed Layer. The original *LambdAgrIoT* architecture uses Apache Sedona, which is a DSMS supporting spatial data within a distributed architecture. However, a more detailed analysis is needed to establish the best system being used. For that reason, we have compared GeoFlink, GeoMesa, and Apache Sedona according to different criteria as detailed in Sect. 6.1. This analysis reveals that *GeoFlink* seems to be the most suitable solution as it excels in real-time processing, which is a priority for detecting anomalies in robot trajectories. Additionally, it offers good scalability and spatial support capabilities, while requiring only moderate resources. Then, we adopt GeoFlink in our new *Speed* layer implementation.

4 Implementation

In this section, we detail the implementation of the continuous queries and the dynamic window approach.

4.1 Esper Extension for Spatial Data

Esper does not support spatial data. Therefore, to implement the *data sources* layer, we have extended the functionalities of Esper. We have implemented a trajectory point data type (i.e., *trajectoryPoint*) in Esper using a Java class.

This class comes with a set of methods that implement spatial operators using HGIS. An example of the spatial operator 'inside a polygon' for the trajectory data point type is shown in Fig. 5. The Esper Java function *IsStreamPointInsidePolygon* takes as a trajectory point data type and the identifier of a polygon. A set of predefined polygons is stored in a table named *Polygons*. The method creates the H2GIS inside spatial query, using the longitude and latitude of the input trajectory point data. Finally, it executes the query. The other spatial operators have been implemented in the same way.

```
public static IsStreamPointInsidePolygon (trajectoryPoint
trajectoryp, integer idpolygon) {

     String query = "SELECT ST_Contains((SELECT geom FROM
                    Polygons
                    WHERE id = idpolygon, " +
                         "ST_GeomFromText('POINT(`|| ? || ' `
                    ||   ? |T ')', 4326))";


     try (PreparedStatement stmt =
          conn.prepareStatement(query)) {
               stmt.setDouble(1, trajectory.getLatitude());
               stmt.setDouble(2, trajectoryp.getLongitude());
     }
     ResultSet rs = stmt.executeQuery();
}
```

Fig. 5. Spatial inside operator.

4.2 Continuous Queries Implementation

In the following we describe the implementation of the continuous queries at the edge, fog, and edge-fog levels.

Continuous queries over robots data can be classified into mono-trajectory queries, involving individual trajectories, and multiple trajectory queries involving a group of trajectories. An example of mono-trajectory query is query **Q1** (Every 5 s, find robot Alpo's top speed in the last 5 s), while query **Q1a** (Every 5 s, find top speed of all robots in the last 5 s) is a multiple-trajectory query.

Another classification, orthogonal to the classification described above, is based on the aggregation functions used:

– A *distributive* aggregation query can be decomposed in sub-(aggregation) queries. For example the query **Q1a** (Every 5 s, find top speed of all robots in the last 5 s) can be computed as follows: (1) compute the max speed for each robot and then (2) select the max of these precomputed values.

– A *non-distributive* aggregation cannot be computed based on partially aggregated values. An example of non-distributive aggregation is query Q2 (every 5 min, compute the total number of robots in a particular geometrical zone of the field within the last minute) since a distinct count aggregation, which is not distributive, must be applied.

Using the above described classifications of the continuous queries, we describe their implementation in the edge, fog and edge-fog levels.

Edge. Queries totally implemented at the edge level are mono-trajectory queries, since they need only data of one robot to being computed. Therefore, they are implemented exclusively using Esper and HGIS.

```
1.  String epl = "select window(*) as points
                  from TrajectoryPointDataType.win:time_batch(5 sec)";
2.  EPStatement statement =
      runtime.getDeploymentService().deploy(epl).getStatements()[0];
3.  statement.addListener((newData, oldData, stat, rt) ->
      isTrajectoryPointInsidePolygon(newData));
```

Fig. 6. Esper implementation of query Q1.

Figure 6 illustrates the implementation of query **Q1** as an example for this class. Every 5 s (i.e., frequency), this query processes all events within the last 5 s (window) using the spatial *IsStreamPointInsidePolygon* method. Esper query language is inspired from SQL with *select* and *from* clauses. The *select*: *window(*)* clause in Line-1 selects all stream data generated by one robot and gives it the alias *points*. In the *from* clause, *TrajectoryPointDataType* Specifies the event type being processed, which is the robot point trajectory. Recall that this class includes the attributes longitude, latitude, speed, and timestamp. The operation *win:time_ batch(30 sec)* sets the window size and the frequency to 30 s. Other kinds of windows exist in Esper. Once the Esper query defined, a listener is attached to this query (Line-2). The *isTrajectoryPointInsidePolygon* is executed each time new data arrive (Line-3).

More complex queries have been implemented at the edge level with Esper and HGIS. For example, the delay query **Q3** (every 5 min, compute the delay of each robot according to their predefined trajectories within the last minute) as shown in Fig. 7a. In order to compute the delay of a robot, it is needed to compare the real time data with the predefined trajectory in this way: compute the distance between the real time GPS position and the planned one, then using the speed compute the time needed to reach planned position from the real one.

In the example of Fig. 7a the robot at time 12:01:20 has a speed of 5 km/h and it is 100 m far from the planned trajectory. Then, its delay is of 72 s.

Fog. Queries implemented exclusively at the fog level are multi-trajectory and non-distributive aggregation queries. This is because their computation need for

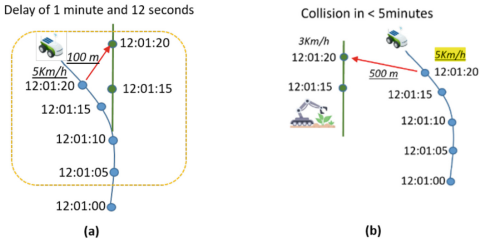

Fig. 7. a) Trajectory delay query example; b) Trajectories collision query example.

all data of different robots, preventing partial computation at the edge. Thus they are implemented at the fog level using only GeoFlink. An example is the collision query **Q4** (Each 10 s, compute if there is a possible collision among all robots) (Fig. 7b). To compute the collision indicator, the distance between pairs of robot trajectories is computed. Then using the maximum speed value, the time for the collision is extrapolated. A threshold is defined in order to trigger a warning or an alert. In the example of Fig. 7b the blue robot has the max speed (5 km/h) and it is 500 m far from the green robot. Thus there is no collision, at least in the next 5 min.

Edge-Fog. Queries that are multi-trajectory queries, thus fog is involved. To being partially computed on the edge level, these queries must involve distributive aggregations. The main idea is to let the robot locally compute a sub-aggregation, and then use the fog level to aggregate these sub-aggregations. An example is the query **Q1a** (every 5 s, compute the max speed of all robots in 5 s). This query can be computed as the max of the result of the local max speed computation of all robots, since the max is a distributive aggregation.

The edge-fog approach can also be used for optimize non-distributive multi-trajectory queries. The main idea is to split the query computation by letting the edge level perform a partial computation that reduce the data sent to the fog for the total aggregation. An example is the optimization of the collision query. To reduce the volume of data that GeoFlink must compute, we have implemented a query optimization technique based on the usage of a buffer zone around the predefined trajectory of each robot. This buffer zone represents a safety area for the robot, where there is no possibility of collisions with the other robots as shown in Fig. 8. Thus, the robot checks that it is within this buffer zone, before sending data to GeoFlink. If this spatial inside computation is positive, then a low frequency is used to transmit data to the fog, as there is no nearby risk of collision. Conversely, when a robot is outside the buffer zone and the error is increasing, a higher frequency is used, resulting in more frequent data transmission from the edge to the fog.

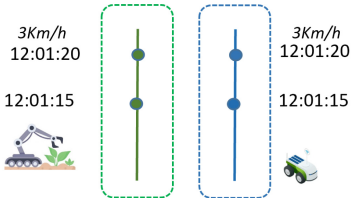

Fig. 8. Collision query optimization at the edge using a safety buffer.

5 Experiments

The edge-fog and the dynamic frequency mechanisms allow for reducing the volume of data sent through the network from the edge to the fog. As counterpart, the fog level receives only a subset of the robotic data, which means that a low frequency simplifies the robots' trajectories. For some simple tests with frequencies 3 s, 15 s, 30 s, we obviously observe that a high frequency does not distort the robot's trajectory, contrary to a low frequency that approximate too much the geometries of the trajectory. More smart frequency sampling approaches can be defined using our dynamic frequency approach, for example when robot maneuver to turn in the field.

In this experiment, we quantify the error according to the frequency used. The error is computed be comparing the distance between the result of the edge-fog query implementation with different frequencies. The experiment is conducted using two machines, one on the robot and the other one at the farm. The farm machine has 32 GB RAM and a quad-core 64-bit i5 processor. The robot machine has 4 GB of RAM and an AMD FX 4100 @3.6 Ghz 64-bit Quad core processor. Trajectory data are sensed by ROS with a frequency of 10 Hz (i.e., 1 data each 0.1 s). We run the query for 30 min, and every 15 s the collision query using 2 robots is triggered. The error is calculated based on MAPE (Mean Absolute Percentage Error).

Figure 9 shows the results of the experiment. It confirms the impact of the frequency on the error. For 15 s the error is under 20%, which is safe for a supervision task. The error decreases to 4% when frequency is high. Thus, we can consider that the collision query can effectively support SA perception in such supervision system.

Next, we compare the query time performance of the proposed edge-fog distribution of the collision query v.s., traditional centralized evaluation at the fog (Fig. 10). Our edge-fog approach is clearly faster than the fog one, thanks to distributed computing and to the reduced data size. The above results show that the edge-fog architecture, including dynamic window resizing mechanisms, query optimization, and in-memory database, has reduced computation time while controlling error, thereby preserving the architecture's reliability.

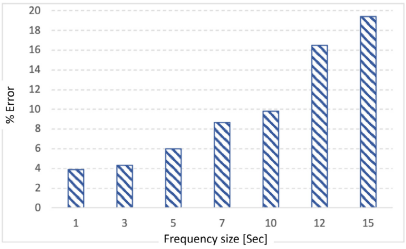

Fig. 9. Edge-fog collision query: Error percentage according to the frequency.

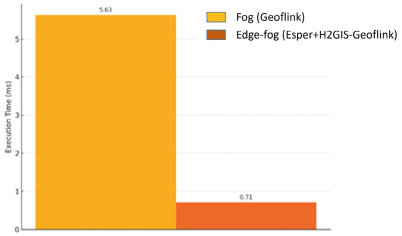

Fig. 10. Collision query: Execution time of edge-fog vs fog approaches.

In the following experiment, we measure how well the edge layer performs, since it plays a key role in helping robots respond in real time. We focus on the processing speed (throughput) of two spatiotemporal functions, using different sliding window sizes, or equivalently different data update rates. The `TrajectoryInsidePolygon` function checks if a robot stays inside a predefined area by making sure that all its location points within a given time window are inside the given polygon. The `Speed` function tracks the robot's average speed in real time. It does this by calculating the speed between each pair of points and then averaging these values over the window. We experiment with the two functions using different observation frequencies ranging from 1 to 15 s. As illustrated in Fig. 11, the cost is clearly affected by the observation frequency (notice the exponential scale on Y-axis). Still, with the highest rate of 15 s, the robot is able to combine the two functions in a query, and respond in less than one second. We believe this throughput is good for the real-time operations of agriculture robots, and that it would even give room for more complex operations and queries.

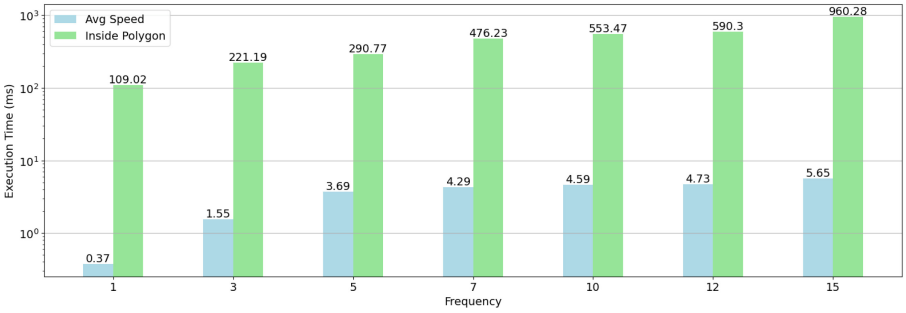

Fig. 11. Execution time of spatial functions in Esper, i.e., in the edge.

6 Related Work

Architectures for Processing Trajectory Data. Agricultural robots, equipped with a variety of sensors, collect vast amounts of spatiotemporal data, including GPS coordinates, speed, and odometry information. This data is streamed in real-time from the robots to be analyzed by a system capable of handling dynamic, high-volume data flows. To manage and analyze this continuous stream of information, a Data Streaming Management System (DSMS) is essential.

A DSMS is designed to efficiently process real-time data streams, making it ideal for agricultural robotics applications. These systems are specifically built to manage large-scale data, support real-time analysis, and execute complex queries on streaming data. In the case of agricultural robots, the trajectory data generated during operations can be sent to the DSMS, which will facilitate advanced analytics, such as detecting anomalies and optimizing performance.

Data Stream Management Systems. DSMS engines are crucial for detecting anomalies in robot trajectories due to their ability to process continuous queries that require scalability and real-time capabilities that traditional databases cannot provide. For this reason, we focus our study on two types of DSMS, namely: (1) *distributed engines* that operate on multiple nodes and (2) *non-distributed engines* that operate on a single node. To assess the compatibility of each solution with the requirements of this project, we highlight the capabilities of these engines in terms of scalability and real-time performance for distributed systems, and lightweight operation for non-distributed systems.

Distributed Engines. For trajectory distributed streaming engines, we have identified eight main solutions that can be used for our project, namely: *GeoFlink*, *Apache Sedona*, *GeoMesa*, *GeoTrellis*, *LocationSpark*, *Magellan*, *NebulaStream*, and *Gelly Streaming*, which are outlined below.

GeoFlink - its main strength is its ability to process real-time spatial data by leveraging *Apache Flink's streaming architecture* and *grid indexing*, enabling efficient continuous queries like range and kNN [20]. Its critical disadvantage is the lack of support for batch processing and persistent storage, which limits its ability to analyze historical spatial data effectively.

Apache Sedona excels at processing large-scale spatial data efficiently in a distributed environment by leveraging *Apache Spark*, with support for batch operations like spatial joins, range queries, and distance calculations. However, it lacks real-time processing capabilities and support for window temporal queries, as it is specifically designed for batch processing workflows [23].

GeoMesa provides storage and query massive spatio-temporal data [6], by leveraging distributed NoSQL databases like *Apache HBase*, *Accumulo*, or *Google Bigtable*. Its main disadvantages include high resource requirements and complex deployment, due to its reliance on these distributed NoSQL databases.

GeoTrellis excels in processing raster data efficiently, but it has limited real-time processing capabilities because it is primarily designed for batch processing

[7]. Additionally, it has limited support for vector data, which restricts its applicability for tasks like robot trajectory analysis.

LocationSpark offers good real-time spatial query capabilities using efficient spatial indexing techniques like R-trees and Quadtrees [22]. However, it has high resource requirements, relies on *Apache Spark's* micro-batch processing architecture, and has been outdated for over 8 years, limiting its applicability for modern real-time applications.

Magellan integrats with *Spark*, enabling batch processing of large-scale vector data [8]. However, it lacks real-time processing capabilities and has limited support for advanced spatial operations such as proximity analysis and buffering.

NebulaStream has high capabilities in processing real-time spatial data for IoT applications [9]. However, its immaturity and complexity in deployment make it an unfavorable choice for large-scale distributed projects.

Gelly Streaming has high real-time capabilities for graph data, but it has limited support for spatial operations and spatio-temporal queries, as it was primarily designed for graph-based data processing [5].

Non-Distributed Engines. Regarding non-distributed streaming engines we identified two main solutions, i.e., *Esper* and *Apache Edgent*.

Esper provides complex event processing (CEP) with advanced support for real-time temporal queries and pattern recognition, as well as its lightweight design and simplicity of deployment, operating with minimal dependencies [3]. Unfortunately, it does not inherently support spatial data, requiring additional tools or custom implementations for geo-spatial processing.

Even though *Apache Edgent* is optimized for streaming data processing on IoT edge devices [4], it has been retired since 2019, and it lacks advanced features like complex event processing (CEP) and pattern recognition. As a consequence, its functionality is limited to less sophisticated real-time data processing tasks (Table 1).

Table 1. Comparison of the fundamental features of *GeoFlink*, *Sedona*, and *GeoMesa*

Feature	GeoFlink	Apache Sedona	GeoMesa
Scalability	High	High	Very High
Real-time capabilities	Very High	Low	Moderate
Spatial support	Moderate	High	Very High
Computational resources	Moderate	Moderate	High

Light-weight Geospatial Databases: in this paper, we needed geospatial processing capabilities at the edge. Due to the resource constraints, traditional database engines would be out of the scope being too heavy, and way beyond the data management at the edge. A vaiable option is thus to use a light-weight in-memory database. Emerging technologies significantly enhance geospatial data management capabilities, particularly for edge computing environments.

DuckDB and *H2GIS* are notable for their lightweight, in-memory architectures that facilitate rapid spatial data processing. DuckDB, known for its embedded OLAP capabilities and spatial extensions, integrates with open-source libraries like GEOS to support complex spatial queries and operations. H2GIS extends the H2 database, providing robust spatial data handling, especially in Java environments [12,19].

Additionally, *Apache GeoParquet*, *Apache Iceberg*, and *Apache IoTDB* offer advanced solutions for managing large-scale spatial datasets across distributed systems. GeoParquet standardizes spatial data storage in an optimized columnar format, while Iceberg facilitates schema evolution and complex data ecosystem management. IoTDB is tailored for time series data in IoT applications, demonstrating significant utility in real-time geospatial processing [1,2,21].

Further alternatives in the landscape, in-process databases such as *Apache Derby*, *Progress OpenEdge*, and *RaimaDB* provide efficient embedded systems with geospatial extensions, suitable for mobile and IoT applications [10,11,16].

7 Conclusion and Future Work

Modern smart farming utilizes autonomous robots and sensors to enable agro-ecological practices. Effective management requires real-time monitoring and control through Data Stream Management Systems (DSMSs). However, the vast volume and potential inaccuracies of robot trajectory data, due to unreliable rural communication networks, pose significant challenges. Existing lightweight DSMSs lack spatial data capabilities, and cloud-fog architectures struggle with poor network conditions. To address this, we propose a hybrid edge-fog architecture and computation framework, building upon our previous work. This framework combines distributed (Geoflink) and extended non-distributed (Esper with spatial operators and dynamic frequency adjustment) DSMSs to efficiently analyze robot trajectory data. A partial computation approach distributes query processing between the edge and fog. Real-world experiments validate the effectiveness of our solution. Our ongoing work concerns the full benchmarking of the proposed solutions. Moreover, we plan to test the usage of different in-memory databases such as DuckDB at the edge level, and benchmark them. Future work is the extension of spatial operators we have implemented in Esper with trajectory operators in order to improve analysis capabilities of our supervision and control system.

Acknowledgements. This work is supported by the French National Research Agency project ANR Ninsar and ANR-24-CHR4-0004-03, French government IDEX-ISITE initiative 16-IDEX-0001 (CAP 20–25) Superob2. Co-author SAKR is partially supported by the EU's Horizon Europe research and innovation programs under Grants No. 101070279 MobiSpaces and No. 101093051 EMERALDS.

References

1. Apache GeoParquet. https://geoparquet.org/. Accessed 21 Mar 2025
2. Apache iceberg. https://iceberg.apache.org. Accessed 21 Mar 2025
3. Complex event processing streaming analytics. https://www.espertech.com/. Accessed Jan 2025
4. Edgent. https://incubator.apache.org/projects/edgent.html. Accessed Jan 2025
5. Gelly streaming. https://github.com/vasia/gelly-streaming. Accessed Jan 2025
6. Geomesa. https://www.geomesa.org/. Accessed Jan 2025
7. Geotrelis. https://geotrellis.io/. Accessed Jan 2025
8. Magellan. https://github.com/harsha2010/magellan. Accessed Jan 2025
9. Nebulastream documentation. https://docs.nebula.stream/. Accessed Jan 2025
10. Progress openedge. https://www.progress.com/openedge. Accessed Mar 2025
11. Raimadb for high-performance embedded databases. https://www.raima.com. Accessed Mar 2025
12. A spatial extension of the h2. http://www.h2gis.org/. Accessed Mar 2025
13. Abdullah, F., Peng, L., Tak, B.: A survey of IoT stream query execution latency optimization within edge and cloud. Wirel. Commun. Mob. Comput. **2021**(1), 4811018 (2021)
14. André, G., et al.: Lambdagriot: a new architecture for agricultural autonomous robots' scheduling: from design to experiments. Cluster Comput. **26**(5), (2023)
15. Bakli, M., Sakr, M., Zimányi, E.: Distributed spatiotemporal trajectory query processing in sql. In: Proceedings of the 28th International Conference on Advances in Geographic Information Systems, pp. 87–98 (2020)
16. Deligiannis, I.: Apache derby: an introduction to security features in an open source database. In: Open Source Systems: Integrating Communities, pp. 158–167. Springer (2016)
17. Dincă, A.-M., Axinte, S.-D., Bacivarov, I.C.: In-memory versus on-disk databases: best practices, use cases and architectural designs. In: 2023 15th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), pp. 1–6. IEEE (2023)
18. Endsley, M.R.: Situation models: an avenue to the modeling of mental models. Proc. Hum. Fact. Ergon. Soc. Ann. Meeting **44**(1), 61–64 (2000)
19. Raasveldt, M., Mühleisen, H.: Duckdb: an embeddable analytical database. In: Proceedings of the 2019 International Conference on Management of Data (2019)
20. Shaikh, S.A., Kitagawa, H., Matono, A., Mariam, K., Kim, K.: Geoflink: an efficient and scalable spatial data stream management system. IEEE Access **10**, (2022)
21. Shuai, X., Wang, J., Zou, J., Tsotras, V.: Apache iotdb: time series database for industrial IoT. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, ACM (2020)
22. Tang, M., Yu, Y., Mahmood, A.R., Malluhi, Q.M., Ouzzani, M., Aref, W.G.: Locationspark: in-memory distributed spatial query processing and optimization. Front. Big Data **3**, (2020)
23. Yu, J., Wu, J., Sarwat, M.: Geospark: a cluster computing framework for processing large-scale spatial data. In: SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM (2015)