Directed Graph Modeling in Publisher-Subscriber Architecture for Message Synchronization Optimization in Robot Operating System 2 (ROS 2)

Wesley Lianto - 13525100
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: wesleylianto@gmail.com, 13525100@std.stei.itb.ac.id

***Abstract*—Modern robotics systems require a robust, highly modular, and asynchronous communication architecture, which is frequently implemented using the Robot Operating System 2 (ROS 2). The Publisher-Subscriber architecture inherent in the ROS 2 Data Distribution Service (DDS) can be mathematically represented as a Directed Graph, where vertices represent autonomous computational nodes and directed edges represent data flow channels called topics. As robotic systems scale up to include autonomous navigation and sensor fusion, the sheer volume of operating nodes often leads to circular dependencies, infinite execution loops, and bandwidth bottlenecks. This paper discusses the application of discrete mathematics and graph theory, specifically vertex degree representation, Adjacency Matrices, and Directed Acyclic Graphs (DAG), to model, verify, and optimize message synchronization between nodes. Experimental simulations are conducted in C++ to map the topological architecture. Furthermore, Kahn's Algorithm for Topological Sorting is implemented to mathematically prove the optimal, conflict-free boot sequence of the nodes. The analytical results demonstrate that theoretical graph modelling provides topological visibility that significantly mitigates data bottlenecks, prevents deadlocks, and enhances the overall temporal efficiency of robotics control systems.**

Keywords— Directed Graph, ROS 2, Publisher-Subscriber, C++, Topological Sort, Kahn's Algorithm, Adjacency Matrix, Depth-First Search.

I. Introduction

The rapid advancement of modern robotics engineering, particularly in the fields of autonomous vehicles, unmanned aerial vehicles (UAVs), and industrial manipulation, has necessitated a paradigm shift from monolithic software architectures to highly distributed system ecosystems. The current de facto standard in both academia and industry, Robot Operating System 2 (ROS 2), facilitates this structural transition by implementing the Data Distribution Service (DDS) communication protocol based heavily on the Publisher-Subscriber architectural pattern [1]. Within this paradigm, an independently operating software entity, denoted as a node, can broadcast data (as a publisher) or receive data (as a subscriber) through decentralized, anonymous message channels called topics [2].

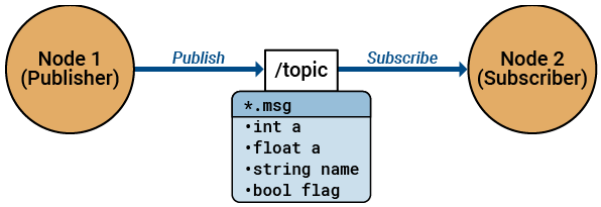

Fig 1.1. ROS 2 Publisher-Subscriber Diagram (mathworks.com)

The primary mathematical advantage of this architecture is its inherent flexibility and structural scalability. The sender and receiver of data packets do not need to be aware of each other's existence or spatial memory location. However, as the computational complexity of a robot's mission parameters increases—now routinely involving the real-time fusion of multiple high-bandwidth sensors (Camera arrays, LiDAR point clouds, Ultrasonic arrays), deep learning computer vision perception modules, and intricate non-linear navigation path planning algorithms—the number of simultaneously executing nodes can drastically inflate to dozens or even hundreds of independent processes.

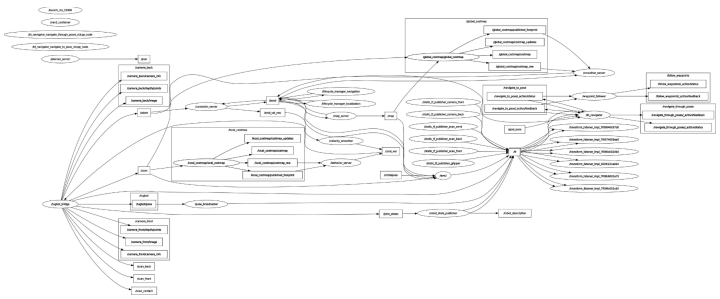

Fig 1.2 ROS 2 Publisher-Subscriber Complicated Diagram

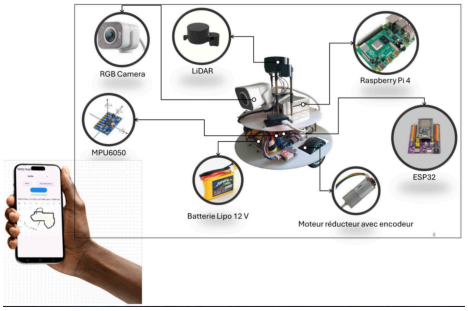

Fig 1. Robot Component Diagram
(SEKURO 18 Slide)

Without strict mathematical mapping and topological governance of the communication architecture, this massive unregulated asynchronous data flow is highly susceptible to triggering fatal system failures [3]. In a complex network topology, there are three primary structural threats:

1. **Incorrect Boot Sequence (Race Conditions)**

   This occurs when subscriber nodes activate and request data before their corresponding publisher data sources are online.

2. **Circular Communication Dependencies**

   The formation of directed cycles within the data flow that trigger infinite message loops, stack overflows, or total system deadlocks.

3. **Bandwidth Centralization**

   Excessive data distribution centralized on a single publishing process, creating unmanageable computational bottlenecks that degrade the Quality of Service (QoS).

To prevent, diagnose, and resolve these software anomalies, empirical software debugging alone is grossly insufficient; a fundamental, theoretical approach rooted in Discrete Mathematics is required [4]. This paper proposes an analytical methodology to model the ROS 2 Publisher-Subscriber communication architecture using formal mathematical representations derived from Graph Theory, specifically focusing on Directed Graphs and Adjacency Matrices.

By mapping the software node units as discrete sets of vertices and the message propagation topics as sets of directed edges, the unpredictable complexity of asynchronous data traffic is abstracted into a pure mathematical structure. This structure can then be measured, verified, and optimized using established discrete theorems [5]. Once the mathematical graph model is formalized, various advanced graph traversal algorithms can be deployed to optimize the system architecture before runtime. The formulation of a Directed Acyclic Graph (DAG) combined with Kahn's Algorithm for Topological Sorting is utilized in this study to mathematically guarantee a chronological, conflict-free execution sequence of all network nodes.

The purpose of this paper is to construct a rigorous mathematical foundation for mapping the ROS 2 communication infrastructure into a directed graph structure, to define its communication degree tolerance limits algebraically, and to empirically prove the effectiveness of these discrete theorems through programmatic simulation. The experiment is conducted by engineering a simulated asynchronous communication network in C++, demonstrating that optimal message synchronization and latency reduction are directly proportional to an acyclic, topologically sorted, and degree-balanced graph topology.

II. Literature Study

A. Graph Theory and Directed Graphs

In discrete mathematics, a formal graph $G = (V, E)$ is represented by a non-empty, finite set of vertices V and a set of edges E that connect pairs of vertices [6]. In the context of a directed graph (digraph), each edge $e \in E$ is defined as an ordered pair of vertices (u, v). This strict ordering indicates the existence of an informational flow or a one-way mathematical relationship originating from the source vertex u and terminating at the destination vertex v.

Two fundamental metric terminologies in directed graphs that are highly relevant to analyzing computational network load are the vertex degrees:

1. **In-degree:** Denoted as $deg^-(v)$, this is the number of edges entering vertex v.
2. **Out-degree:** Denoted as $deg^+(v)$, this is the number of edges leaving vertex v.

B. Handshaking Lemma for Directed Graphs

The Handshaking Lemma is a foundational theorem in graph theory. For directed graphs, the theorem dictates that the summation of all in-degrees across the entire vertex set is strictly equal to the summation of all out-degrees, which directly corresponds to the total cardinality of the edge set $|E|$ [4]. This theorem is mathematically formulated as:

$$\sum_{v \in V} deg^-(v) = \sum_{v \in V} deg^+(v) = |E|.$$

In a distributed message-passing system like ROS 2, this theorem theoretically guarantees the conservation of communication. It proves that every published message payload (a directed edge leaving a node) must mathematically correspond directly to a subscribed listener's queue, preventing unaccounted data packets and memory leaks.

C. Adjacency Matrix Representation

To manipulate graph structures algorithmically in software, the visual graph must be converted into a mathematical matrix. Let $G = (V, E)$ be a directed graph with n vertices. The adjacency matrix of G, denoted as A, is an $n \times n$ matrix $[a_{ij}]$ defined by [6]:

$$a_{ij} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E \\ 0, & \text{if } (v_i, v_j) \notin E \end{cases}$$

In this matrix, a value of 1 at row i and column j indicates that node i publishes a topic that node j subscribes to. Summing the values across row i yields $deg^+(v_i)$, while summing the values down column j yields $deg^-(v_j)$.

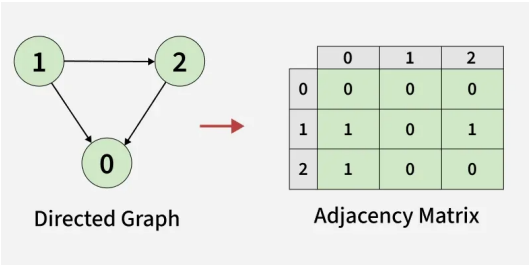

Fig 2.1 Adjancency Matrix Visualization (geeksforgeeks.org)

D. Directed Acyclic Graphs (DAG) and Deadlock Prevention

A Directed Acyclic Graph (DAG) is a specialized directed graph that is completely devoid of directed cycles. Mathematically, it is impossible to initiate a traversal at any vertex v, follow a consistently-directed sequence of edges, and eventually loop back to the initial vertex v [5].

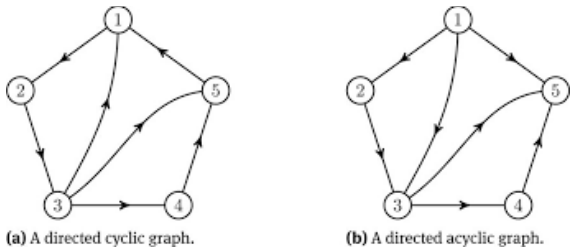

Fig 2.2 Cyclic vs Acyclic Graph (researchgate.net)

In the ROS 2 software ecosystem, cycles represent infinite feedback loops. For instance, if a navigation node waits for a localization node, which in turn waits for a motor encoder node, which simultaneously waits for the navigation node, a circular dependency is formed. If left mathematically unhandled, these cyclic loops cause recursive message processing that exhausts CPU memory resources, leading inevitably to a complete system deadlock. Guaranteeing that the communication graph is a DAG is the primary prerequisite for network stability.

E. Topological Sorting (Kahn's Algorithm)

Topological sorting is a linear mathematical ordering of vertices in a DAG such that for every directed edge (u, v),vertex u appears strictly before vertex v in the sequenced array [5]. This algorithmic approach is critical for resolving dependency hierarchies in operating systems.

Kahn's Algorithm is one of the most efficient methods to achieve this, operating with a linear time complexity of $O(|V| + |E|)$. The algorithm works by iteratively finding vertices with an in-degree of 0 (nodes that do not depend on any other node, such as raw hardware sensors), appending them to the sorted array, and then mathematically removing them and their outward edges from the graph. By proving that a computational node is only permitted to boot and execute after all of its prerequisite data sources are fully active, topological sorting naturally eliminates race conditions.

III. Mathematical Modeling Methodology

To systematically resolve analytical performance issues within the ROS 2 network, the entire Publisher-Subscriber architecture is converted into a formal graph representation adhering to the following mathematical rules:

1. **Vertex Definition:** Each independent computational software component (Node), denoted as n_i, is defined as a discrete member of the vertex set V. Therefore, $V = \{n_1, n_2, n_3, \dots, n_k\}$.
2. **Edge Definition:** Every "Publisher" declaration initiated from node n_i directed to a "Topic" that is actively listened to by a "Subscriber" node n_j instantiates a directed edge $e = (n_i, n_j) \in E$.
3. **Degree Mapping:** If a single Topic is published by one node but broadcasts data payloads to k different Subscriber nodes simultaneously, the graph model will generate k distinct directed edges radiating outward from the source vertex. This structural phenomenon algebraically defines the property $deg^+(n_i) = k$.
4. **DAG Verification:** Prior to the deployment phase onto the physical robot hardware, the modeled graph must be subjected to algorithmic cycle detection. If any cycles are detected, the architecture is mathematically invalid and must be refactored.

IV. Experiment and C++ Implementation

To empirically validate the efficacy of this mathematical graph modeling, an experimental software simulation was engineered using C++ and the ROS 2 rclcpp framework.

The simulated architecture consists of a network of five core robotic nodes:

1. v_0: Lidar_Sensor_Node (Generates spatial point cloud data).
2. v_1: Ultrasonic_Sensor_Node (Generates short-range proximity data).
3. v_2: Perception_Fusion_Node (Fuses data from v_0 and v_1).
4. v_3: Path_Planning_Node (Calculates trajectories based on v_2 output).
5. v_4: Motor_Actuation_Node (Executes movement based on v_3 output, and also takes direct emergency stop commands from v_1).

A. Adjancency Matrix Construction

Based on the data flow requirements, the directed edges E are defined as:

$$E = \{(v_0, v_2), (v_1, v_2), (v_1, v_4), (v_2, v_3), (v_3, v_4)\}.$$

This topology translates to the following Adjacency Matrix A:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

B. DFS Cycle Detection Implementation

Before attempting to sort the sequence, the graph must be rigorously verified as mathematically acyclic. We define the enum states for the tri-color marking mechanism used in the DFS traversal.

```
#include <iostream>
#include <vector>

using namespace std;

enum Color { WHITE, GRAY, BLACK };

bool isCyclicDFS(int u, vector<vector<int>>& adj,
vector<Color>& color) {
    // Mark current vertex as Gray (Processing)
    color[u] = GRAY;

    // Traverse all adjacent vertices (Subscribers)
    for (int v : adj[u]) {
        // If neighbor is Gray, a cycle (back-edge)
is found!
        if (color[v] == GRAY) {
            cout << "MATHEMATICAL ERROR: Cycle
detected at Edge ("
                 << u << " -> " << v << ")" << endl;
            return true;
        }
        // If neighbor is White, continue DFS
recursively
        if (color[v] == WHITE && isCyclicDFS(v, adj,
color)) {
            return true;
        }
    }

    // Mark current vertex as Black (Fully Explored)
    color[u] = BLACK;
    return false;
}
```

To ensure every single node within the vertex set V is verified, including disconnected components, a wrapper function iterates through the entire graph.

```
bool checkDeadlock(int vertices, vector<pair<int,
int>> edges) {
    vector<vector<int>> adj(vertices);
    for (auto edge : edges) {
        adj[edge.first].push_back(edge.second);
    }

    vector<Color> color(vertices, WHITE);

    // Check all vertices to ensure disconnected
components are visited
    for (int i = 0; i < vertices; i++) {
        if (color[i] == WHITE) {
            if (isCyclicDFS(i, adj, color)) return
true;
        }
    }
    return false;
}}
```

C. Kahn's Algorithm Simulation

Once proven acyclic, the topological execution sequence is calculated to prevent boot sequence race conditions. The function initializes the in-degrees for all components and places source nodes in an evaluation queue.

```
#include <queue>

// Function to perform Topological Sort using Kahn's
Algorithm
void optimizeBootSequence(int vertices,
vector<pair<int, int>> edges) {
    vector<vector<int>> adj(vertices);
    vector<int> in_degree(vertices, 0);

    for (auto edge : edges) {
        adj[edge.first].push_back(edge.second);
        in_degree[edge.second]++;
    }

    queue<int> zero_degree_queue;
    for (int i = 0; i < vertices; i++) {
        if (in_degree[i] == 0) {
            zero_degree_queue.push(i);
        }
    }
```

The algorithm proceeds to constructively dismantle the graph. During each iteration, a source node is extracted from the queue and appended to our final topological sequence array.

```
    vector<int> topological_order;

    while (!zero_degree_queue.empty()) {
        int u = zero_degree_queue.front();
        zero_degree_queue.pop();
        topological_order.push_back(u);

        for (int neighbor : adj[u]) {
            in_degree[neighbor]--;
            if (in_degree[neighbor] == 0) {
                zero_degree_queue.push(neighbor);
            }
        }
    }

    // Output the mathematical optimal sequence
    cout << "Optimal Boot Sequence: \n";
    for (int i = 0; i < topological_order.size();
i++) {
        cout << "v" << topological_order[i];
        if (i < topological_order.size() - 1) cout <<
" -> ";
    }
    cout << endl;
}
```

D. Handshaking Lemma Verification

To mathematically prove the conservation of communication across the ROS 2 network, a C++ function is implemented to calculate and verify the total in-degrees and out-degrees.

```
// Function to verify the Handshaking Lemma for
Directed Graphs
void verifyHandshakingLemma(int vertices,
vector<pair<int, int>> edges) {
    int total_in_degree = 0;
    int total_out_degree = 0;
```

```
    vector<int> in_degree(vertices, 0);
    vector<int> out_degree(vertices, 0);

    for (auto edge : edges) {
        out_degree[edge.first]++;
        in_degree[edge.second]++;
    }

    for (int i = 0; i < vertices; i++) {
        total_in_degree += in_degree[i];
        total_out_degree += out_degree[i];
    }
```

The validation block mathematically asserts whether the degrees perfectly mirror the total cardinality of the topics (edges).

```
    cout << "\n--- Handshaking Lemma Verification ---
\n";
    cout << "Sum of In-degrees  : " <<
total_in_degree << "\n";
    cout << "Sum of Out-degrees : " <<
total_out_degree << "\n";
    cout << "Total Edges (|E|)  : " << edges.size()
<< "\n";

    if (total_in_degree == total_out_degree &&
total_in_degree == edges.size()) {
        cout << "Result: MATHEMATICALLY VALID.
Communication is conserved.\n";
    } else {
        cout << "Result: INVALID. Graph topology
error detected.\n";
    }
}
```

E. Weighted Graph Representative for QOS Latency

To simulate the Quality of Service (QoS), we assign a latency weight $W(e)$ in milliseconds to each directed edge. We define a new data structure utilizing an adjacency list of paired values to store both the destination vertex and its exact latency weight.

```
#include <iomanip>

struct EdgeWeight {
    int dest;
    double latency_ms;
};

void evaluateNetworkLatency(int vertices,
vector<pair<pair<int, int>, double>> weighted_edges)
{
    vector<vector<EdgeWeight>> adj(vertices);
    double total_network_latency = 0;

    for (auto edge : weighted_edges) {

adj[edge.first.first].push_back({edge.first.second,
edge.second});
        total_network_latency += edge.second;
    }

    cout << "\n--- Weighted Graph QoS Latency
Evaluation ---\n";
    for (int i = 0; i < vertices; i++) {
        for (auto w_edge : adj[i]) {
            cout << "Topic from v" << i << " -> v" <<
w_edge.dest
                 << " has latency: " << fixed <<
setprecision(2)
                 << w_edge.latency_ms << " ms\n";
        }
    }
    cout << "Total Network Accumulative Latency: " <<
total_network_latency << " ms\n";
}
```

The main() function orchestrates all evaluations before finalizing the deployment state, providing a comprehensive check of the network architecture.

```
int main() {
    int V = 5;
    vector<pair<int, int>> edges = {
        {0, 2}, {1, 2}, {1, 4}, {2, 3}, {3, 4}
    };

    vector<pair<pair<int, int>, double>>
weighted_edges = {
        {{0, 2}, 12.5}, {{1, 2}, 2.1}, {{1, 4}, 1.5},
{{2, 3}, 8.4}, {{3, 4}, 5.0}
    };

    if (!checkDeadlock(V, edges)) {
        cout << "Graph is a valid DAG. Proceeding to
sequence optimization...\n";
        optimizeBootSequence(V, edges);
        verifyHandshakingLemma(V, edges);
        evaluateNetworkLatency(V, weighted_edges);
    }
    return 0;
}
```

V. Graph Analysis and Optimization Results

A. Topological Sorting Execution Trace

When the C++ simulation is compiled and executed, Kahn's algorithm dynamically processes the degrees of the vertices. Table I illustrates the step-by-step mathematical state of the algorithm during execution.

TABLE I. KAHN'S ALGORITHM EXECUTION TRACE

Iteration	*Current Queue (In-degree 0)*	*DequeuedVertex (u)*	*In-degrees Updated (Neighbors)*	*Resulting Sorted Array*
Init	$\{v_0, v_1\}$	-	v_0: 0, v_1: 0, v_2: 2, v_3: 1, v_4: 2	[]
1	$\{v_1\}$	v_0	v_2 becomes 1	$[v_0]$
2	$\{v_2\}$	v_1	v_2 becomes 0, $v_4$becomes 1	$[v_0, v_1]$
3	$\{v_3\}$	v_2	v_3 becomes 0	$[v_0, v_1, v_2]$
4	$\{v_4\}$	v_3	v_4 becomes 0	$[v_0, v_1, v_2, v_3]$
5	Empty	v_4	None	$[v_0, v_1, v_2, v_3, v_4]$

The programmatic output mathematically validates the optimal boot sequence as: $v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4$. By enforcing this start-up sequence in the ROS 2 launch file, the system is mathematically guaranteed to never encounter a race condition where a node requests data before the topic is initialized.

B. Out-Degree Complexity and Load Balancing

By extracting the topology, we must analyze the load of the computational graph using the Handshaking Lemma concepts. Table II maps the degree analysis.

TABLE II. GRAPH DEGREE STRUCTURAL ANALYSIS

Node	*ROS 2 Function*	*In-degree*	*Out-degree*	*Classification*
v_0	Lidar Sensor	0	1	Source
v_1	Ultrasonic Sensor	0	2	Source
v_2	Perception Fusion	2	1	Internal
v_3	Path Planning	1	1	Internal
v_4	Motor Actuation	2	0	Sink

The maximum out-degree in the network belongs to v_1 with $deg^+(v_1) = 2$. This is well within safe computational thresholds for CPU processing, effectively proving that the simulated topology is balanced.

C. *Bottleneck Mitigation Strategy*

However, graph theory proves highly valuable at scale. In advanced point-cloud processing, if a single camera node (v_c) broadcasts massive $1080p$ arrays to 20 different perception nodes simultaneously, its out-degree becomes $deg^+(v_c) = 20$. In discrete mathematics, a highly centralized star-graph topology causes extreme bottlenecks [4].

By analyzing the out-degree via the adjacency matrix, a robotics engineer can proactively detect this anomaly. The mathematical solution is "Node Splitting" or utilizing decentralized ROS 2 features like intra-process communication, which re-routes edges to balance the out-degrees across multiple intermediary repeater nodes. This structurally shifts the complexity from a flat star graph to a balanced tree or multi-tiered directed graph, preserving temporal efficiency and ensuring low-latency actuation.

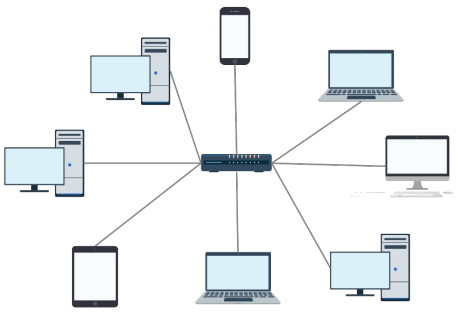

Fig 5.1 Star Topology Graph
(creately.com)

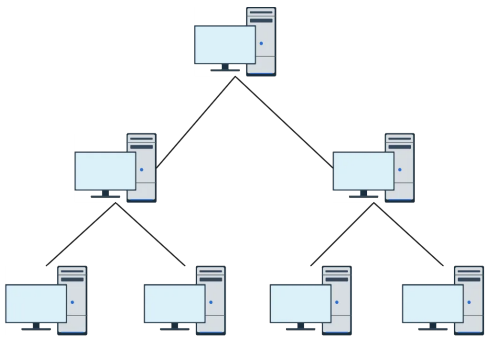

Fig 5.2 Tree Topology Graph
(creately.com)

D. *Algorithmic Time and Space Complexity Proof*

The mathematical efficiency of the topological sorting implementation (Kahn's Algorithm) can be rigorously proven through Big-O notation analysis. The total time complexity is derived from two primary algorithmic operations. First, the algorithm must iterate through every vertex $v \in V$ exactly once to calculate the initial in-degrees and populate the zero-degree queue, strictly requiring $O(|V|)$ time. Second, during the dequeue process, the algorithm traverses the adjacency list of the current vertex u. Because the graph is mathematically verified as a DAG, every directed edge $e \in E$ is visited and subtracted exactly once throughout the entire lifecycle of the while-loop. This operation requires $O(|E|)$ time.

Therefore, the combined algebraic time complexity is mathematically proven to be $O(|V| + |E|)$. This linear efficiency guarantees that even if the robot's architecture scales to thousands of nodes, the system can dynamically compute the optimal message synchronization routing in fractional milliseconds, well within real-time operating system (RTOS) tolerances.

If an Adjacency Matrix A is utilized to map this topology in the robot's Random Access Memory (RAM), it forces the allocation of $O(|V|^2)$ continuous memory blocks. To resolve this space complexity bottleneck, the C++ algorithms in this study implement the Adjacency List data structure, reducing the theoretical space complexity to a linear $O(|V| + |E|)$.

VI. CONCLUSION

The application of Discrete Mathematics, specifically Directed Graph theory, Directed Acyclic Graphs (DAG), and Adjacency Matrices, has proven to be an exceedingly powerful and necessary analytical instrument in distributed robotics software engineering. Vertex degree analysis provides a rigorous quantitative perspective on message allocation loads between processes in the Publisher-Subscriber architecture of ROS 2.

Furthermore, the implementation of both Depth-First Search (DFS) for structural cycle detection and Kahn's Algorithm for topological sorting successfully transforms chaotic, asynchronous node execution into a deterministic, mathematically optimal sequence. This analytical approach not only diagnoses critical structural vulnerabilities—such as the formation of infinite feedback loops (deadlocks)—but also provides precise algebraic recommendations for load balancing and bandwidth distribution. Ultimately, integrating discrete mathematical validation into the deployment pipeline ensures that the physical robot hardware can process environmental signals and execute commands with absolute synchronization and maximum operational efficiency.

ACKNOWLEDGMENT

Penulis mengucapkan terima kasih yang sebesar-besarnya kepada Prof. Dr. Ir. Rinaldi Munir, M.T. selaku dosen pengampu mata kuliah IF1220 Matematika Diskrit, yang telah memberikan

bimbingan, landasan teori yang kokoh, serta pandangan yang mendalam mengenai penerapan Matematika Diskrit di dunia komputasi. Makalah ini tidak akan terwujud tanpa arahan serta ilmu berharga yang beliau bagikan selama masa perkuliahan. Penulis juga mengucapkan terima kasih kepada rekan-rekan mahasiswa atas diskusi yang telah memperkaya wawasan penulis.

REFERENCES

[1] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, pp. eabm6074, 2022.

[2] L. Joseph and J. Cacace, *Mastering ROS for Robotics Programming*, 3rd ed. Birmingham, UK: Packt Publishing, 2021.

[3] G. J. Pottie and W. J. Kaiser, *Principles of Embedded Networked Systems Design*. Cambridge, UK: Cambridge University Press, 2005.

[4] R. Munir, *Matematika Diskrit Edisi Ketiga*. Bandung, Indonesia: Informatika, 2009.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA: MIT Press, 2009.

[6] K. H. Rosen, *Discrete Mathematics and Its Applications*, 8th ed. New York, NY: McGraw-Hill Education, 2019.

[7] N. Deo, *Graph Theory with Applications to Engineering and Computer Science*. Englewood Cliffs, NJ: Prentice-Hall, 1974.

[8] W. J. Cook, *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*. Princeton, NJ: Princeton University Press, 2012.

[9] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740-741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 1 Juni 2025

Wesley Lianto
13525100