electronics

Article

An Effective Method for Constructing a Robot Operating System Node Knowledge Graph Based on Open-Source Robotics Repositories

Yuxin Zhao [1], Xinjun Mao [1,*] and Yi Yang [2]

[1] College of Computer, National University of Defense Technology, Changsha 410073, China; yuxinzhao@nudt.edu.cn
[2] College of Information Science and Engineering, Hunan Women's University, Changsha 410116, China; snryou@126.com
* Correspondence: xjmao@nudt.edu.cn

Abstract: Robot software development can be considered as a component-driven process, and existing ROS components, such as an ROS node, can be reused to construct robot applications. By reusing the ROS node, the development process of robot software can be significantly accelerated. However, the challenges in reusing ROS nodes primarily lie in the scattered organization of ROS node information. To address this challenge, this paper proposes a method to construct an ROS node knowledge graph (RNKG) based on high-quality open-source robot projects. In order to build a high-quality knowledge graph of ROS nodes, we first constructed a high-quality dataset of open-source robot projects. Since ROS node knowledge can exist in both text and code formats, we initially separated the data in the dataset into code data and text data, and then applied different knowledge extraction methods to extract corresponding entities. Finally, we integrated a series of ROS node knowledge and organized it into a knowledge graph. To validate the effectiveness of the constructed ROS node knowledge graph, we first verified the completeness of the entities and the accuracy of relationships in the knowledge graph. Next, we evaluated the performance of the ROS node knowledge graph in assisting developers with the downstream task of finding ROS nodes. These findings suggest that our proposed method for constructing an ROS node knowledge graph is feasible and demonstrate that the ROS node knowledge graph helps search ROS nodes.

Keywords: knowledge graph; ROS nodes; robotic software development

Citation: Zhao, Y.; Mao, X.; Yang, Y. An Effective Method for Constructing a Robot Operating System Node Knowledge Graph Based on Open-Source Robotics Repositories. *Electronics* **2023**, *12*, 4022. https://doi.org/10.3390/electronics12194022

Academic Editor: Ashwin Ashok

Received: 14 August 2023
Revised: 15 September 2023
Accepted: 18 September 2023
Published: 24 September 2023

1. Introduction

The field of robot software development has witnessed significant progress. A robot operating system (ROS) [1] is the most widely adopted platform for developing robot software. ROS nodes serve as a fundamental component throughout the process of robot software development, and developers can leverage ROS nodes to construct complex and sophisticated robot software systems [2]. The ROS community and GitHub offer a wealth of open-source ROS nodes with diverse functionalities. A knowledge graph can serve as a shared and centralized database [3] to store a vast amount of ROS node's knowledge. The ROS node knowledge graph can help ROS developers develop robot software by rapidly reusing existing ROS node components [4,5]. This method can reduce the development time and workload required for robot software.

The challenges of reusing ROS nodes primarily lie in the scattered organization of ROS node information, the complexity and dependencies among ROS nodes, and the difficulty comprehending ROS node code. For the first challenge, scattered ROS node knowledge arises from the lack of a universal standard or website for organizing and categorizing ROS nodes, resulting in diverse contributions by scholars and institutions [6]. For the second challenge, the ROS ecosystem encompasses thousands of nodes with complex dependency

relationships and communication mechanisms. Gathering and organizing information about these nodes becomes highly difficult and time-consuming [7]. For the last challenge, the lack of consistent documentation and examples in ROS node code, which different developers often write, poses a significant challenge in understanding and maintaining the code.

Based on the above challenges in constructing an ROS node knowledge graph, we propose a method for producing an ROS node knowledge graph based on high-quality open-source robotics repositories. To accomplish knowledge graph building, we started by creating a reliable database of robot projects sourced from open-source code platforms. As various types of resources necessitate different knowledge extraction methods, we categorized the ROS node resources discovered within these open-source projects. Moving on to the second stage, we employed distinct approaches to extract ROS node entities for different types of resources. Firstly, we utilized code snippet extraction rules to extract the required code segments from a vast collection of ROS node code, thereby enhancing the accuracy of ROS node entity identification. We extracted the corresponding ROS node entities from the code snippets by applying various code parsing rules. Secondly, we employed part-of-speech tagging to extract functional and hardware knowledge mentioned in the ROS node description statements for the textual descriptions of ROS nodes. We then classified the functionality and hardware of each ROS node based on predefined rules. Finally, in the last stage, we gathered the data within the ROS node knowledge graph and successfully constructed the knowledge graph for the ROS node.

The specific contributions of this paper can be summarized as follows:

- We constructed an ROS domain dataset from high-quality open-source robotics repositories. Using open-source robotics projects, we proposed a systematic method to build the ROS node knowledge graph;
- We constructed RNKG: a domain knowledge graph that effectively organizes scattered knowledge of ROS nodes and accurately represents the semantics of ROS nodes through the defined entities and relationships;
- We validated entities' completeness and relationship accuracy in RNKG, demonstrating its effectiveness. Furthermore, we provide evidence that RNKG can assist in robotic software development through an ROS node search task.

The remainder of this paper is organized as follows. Section 2 introduces the related work. Section 3 describes our methods. Evaluations are provided in Section 4. Finally, we conclude the paper in Section 5.

2. Related Work

This section briefly summarizes the related studies of ROS component reuse and the research status of knowledge graphs in robotics.

2.1. ROS Component Reuse

The ROS is critical in developing robotics software by offering a diverse collection of reusable components. These components, including ROS packages, ROS nodes, ROS messages, ROS services, and ROS actions, serve as fundamental building blocks for creating and implementing complex robot applications. By leveraging these reusable components, developers benefit from modularity, code reusability, and scalability throughout the robotics software development process. The ROS community also plays a vital role in supporting the utilization of these components. It provides developers access to numerous public code libraries and tools contributed by the ROS community. These resources facilitate component reuse and contribute to a rich functionalities, algorithms, and tools ecosystem. Developers can leverage this ecosystem to accelerate their development and build more advanced robot applications.

ROS packages [8,9] serve as practical containers for organizing and distributing related functionalities within a robotics system. They provide a convenient way to package and manage software components in ROS. ROS nodes [10,11] are individual processes that

carry out specific tasks within a robot's system. They can be reused in different robotics software to achieve similar functions, promoting code reusability and modularity. By utilizing ROS community-contributed messages [12], developers gain access to a wide range of functionalities and data formats. This fosters collaboration and knowledge sharing within the ROS community, allowing developers to leverage existing solutions and easily exchange data between different systems. Reusing ROS services [13] enables developers to build modular and scalable robot applications. It facilitates distributed computing and leverages standardized interfaces, ensuring consistent and interoperable communication between software components.

In summary, an ROS and its reusable components, supported by the ROS community, optimize the reuse of ROS nodes and facilitate robotics software development. They offer modularity, code reusability, and standardized interfaces, ultimately enhancing efficiency and promoting collaborative development in robotics.

2.2. Robotics Domain Knowledge Graph Application

Due to the advanced knowledge reasoning and recommendation capabilities of knowledge graphs, they have been extensively investigated in robotics, leading to significant advancements. Integrating knowledge graphs into the knowledge reasoning process enhances robots' ability to analyze and process environmental information [14]. Moreover, knowledge graphs play a crucial role in driving robot software development through their recommendation power. Specifically, in the context of an ROS (robot operating system), the ROS-related knowledge graph can facilitate tasks such as searching for ROS packages [8] and ROS messages [12], simplifying the software development process for robotics.

In the research process, the knowledge reasoning ability and recommendation searchability of the knowledge graph will help ROS developers find the ROS node they need.

3. Methodology

To collect scattered ROS node knowledge and employ a standardized approach using a knowledge graph, the process is outlined in Figure 1 and is divided into three stages. The first stage is dataset construction and resource classification, followed by the second stage of further processing and entity extraction from the obtained resources. The final stage involves knowledge graph construction for the ROS node knowledge graph.

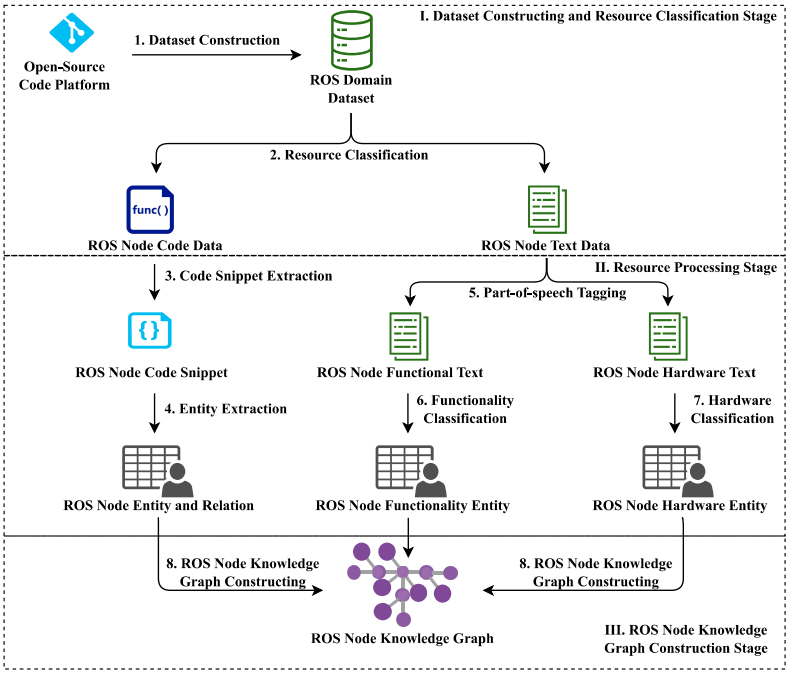

Figure 1. The process of constructing ROS node knowledge graph.

In the first stage, we built a high-quality dataset of robot projects from open-source code platforms. Then, considering different types of resources require distinct knowledge extraction methods, we classified the ROS node resources found in the open-source projects. In the second stage, we adopted different approaches to extract ROS node entities for various resource types. Firstly, for ROS node code, we applied code snippet extraction rules to extract the necessary code segments from a vast collection of ROS node code, which enhanced the accuracy of ROS node entity identification. Subsequently, using different code parsing rules, we extracted the corresponding ROS node entities from the code snippets. Secondly, we employed part-of-speech tagging to extract the functional knowledge and hardware knowledge mentioned in the ROS node description statements for the textual descriptions of ROS nodes. Then, based on predefined applicable and hardware rules, we classified the functionality and hardware of each ROS node. In the final stage, we collected the data in the ROS node knowledge graph and ultimately succeeded in constructing the ROS node knowledge graph.

3.1. Dataset Constructing and Processing Stage

The dataset construction and dataset processing stages consist of two substeps. The first step involves building the robot software project dataset from open-source platforms. In contrast, the second step involves processing the constructed robot software dataset to facilitate subsequent entity and relationship extraction.

3.1.1. Dataset Construction

The purpose of constructing the ROS node knowledge graph is to integrate dispersed ROS node knowledge and support robot software development. While the ROS Wiki is the predominant resource-sharing platform in the ROS community, its coverage of ROS node knowledge is not comprehensive. Therefore, we needed gather extensive ROS node knowledge from other data sources. Since the ROS node knowledge contained in ROS Wiki can be found in ROS software packages available on open-source platforms, we were able to utilize robot software projects provided by these platforms as a data source for constructing the ROS node knowledge graph.

Ivano et al. [15] provided a high-quality dataset of robot software projects collected from open-source platforms using a scientific approach. This dataset includes 598 robot software projects that underwent manual quality checks and represent high-quality projects in the real world. However, to make this dataset suitable for building the ROS node knowledge graph, we manually analyzed each robot project and applied a set of systematic exclusion criteria for filtering. Table 1 shows the specific filtering rules.

Table 1. Criteria for filtering robot open-source projects aimed at constructing the ROS node knowledge graph.

ID	Description of the Criteria for Filtering Robot Open-Source Projects
R1	Assume that a robot software project is related to a robot code development platform, then exclude the project.
R2	Assume that a robot software project only contains launch files without the src and scripts directories, then exclude the project.
R3	Assume that the primary programming language of a robot software project is not C++ or Python, then exclude the project.
R4	Assume that a robot project has been deprecated (as of December 2022), then exclude the project.
R5	Assume that the core functionality of a robot software project is to simulate robots or a simulated robot environment, then exclude the project.
R6	Assume that a robot software project provides development tools within ROS, such as genius, and rqt bag, then exclude the project.
R7	Assume that the purpose of a robot software project is to help developers learn ROS or practice ROS-related tasks, then exclude the project.

Any robot software project that meets any of the filtering criteria was discarded, and the remaining robot software projects were retained as a dataset for building the ROS node knowledge graph. After filtering, we ended up with a total of 503 robot software projects. The results are shown in Table 2, where we can observe the basic statistical results of the robot software projects. These data clearly demonstrate that our dataset is of high-quality, and the selected robot projects can represent the real-world robot software development process.

Table 2. Descriptive statistical results of the open-source robot project dataset (SD = standard deviation, CV = coefficient of variation).

	Min.	Max.	Median	Mean	SD	CV
Commits	100	7611	248	533.99	855.36	1.60
Contributors	1	233	12	19.11	24.45	1.28
Branches	2	483	6	9.78	24.65	2.52
Issues	0	983	13	48.79	101.35	2.07
Pull requests	0	2165	16	62.80	163.46	2.60
Launch files	0	579	11	26.09	48.70	1.86

3.1.2. Dataset Processing

Robot software projects are complex and challenging to handle. To improve the accuracy of ROS node entity extraction and relationship extraction, we processed the constructed robot project dataset and divided them into two categories: ROS node code data and ROS node textual data.

A robot project mainly includes CMakeLists.txt, package.xml, launch files, config directory, src directory, include directory, scripts directory, message directory, service directory, urdf directory, world directory, and README file. The process of distinguishing the open-source robot project into ROS node code data and ROS node file data can be found in Figure 2. In order to obtain as much ROS node textual data as possible, we considered the comments in the src directory, and included directory, scripts directory, and the text information in the README file as textual data of the ROS nodes. We considered the src directory, include directory, scripts directory, message directory, service directory, and launch files in the robot project as ROS node code data.

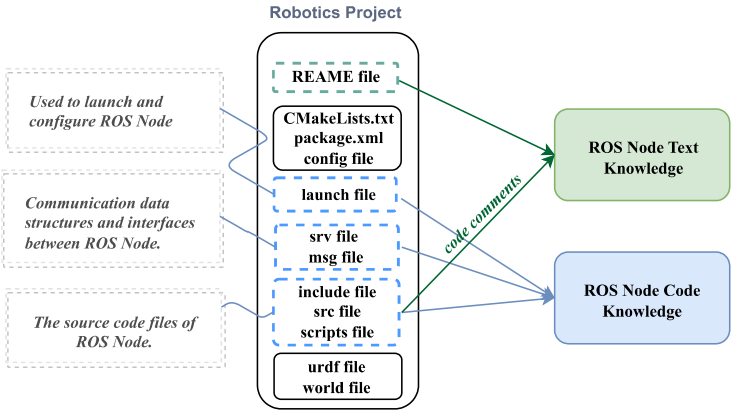

Figure 2. The process and reason for distinguishing the open-source robot project into ROS node code data and ROS node file data.

3.2. ROS Node Resource Processing Stage

The ROS node resource processing phase is divided into four subprocesses. The first two subprocesses involve extracting ROS node entities and relationships from the ROS node code data. The third subprocess involves extracting function-related entities from the ROS node text data. The final subprocess involves extracting hardware-related entities from

the ROS node text data. These four processes perform relationship and entity extraction of ROS nodes from their respective datasets.

3.2.1. Code Snippet Extraction

To extract the required ROS node entities and relationships from ROS node code data, we created different rule templates to extract the ROS node code knowledge. Table 3 presents the rules for extracting ROS node code snippet data from C++ files. Whenever a code snippet met the corresponding extraction rule, we retained that code snippet for subsequent entity and relationship extraction. Following this approach, we were able to further design regulations and reasons for extracting Python code snippets.

Table 3. ROS code snippet extraction rules and reasons (C++).

ROS Node Code Snippet Data Extraction Rules	The Reason for Extracting ROS Node Code Snippet
Extract code segments containing "ros::init"	"ros::init" defines the name of the ROS node.
Extract code segments containing "nh.advertise"	"nh.advertise" defines the ROS node knowledge related to publishing topics.
Extract code segments containing "nh.subscribe"	"nh.subscribe" defines the ROS node knowledge related to subscribing to topics.
Extract code segments containing "nh.serviceClient"	"nh.serviceClient" defines the outgoing service requests in the ROS node.
Extract code segments containing "nh.advertiseService".	"nh.advertiseService" defines the services provided by other ROS node.
Extract all method definitions in the ROS node code	A portion of the method definitions are callback functions.
Extract parameter settings in ROS node code	A portion of the parameter definitions includes topic names and message types.

3.2.2. Entity Extraction and Relation Extraction

After extracting code snippets related to ROS nodes, we must define extraction methods for ROS node entities in Figures 3 and 4. These methods guide us to extract the necessary entities and relationships from the code snippets to build an ROS node knowledge graph.

The code snippet entity extraction method shown in Figure 3 primarily focuses on extracting entities related to the 'advertise' topic and the 'needed service'. In this case, if the field/name refers to 'advertise', the value/arguments correspond to the topic name, and the field/arguments correspond to the ROS message. Different relationships are then built for these entities based on the field/name. If the field/name refers to 'ServiceClient', the value/arguments correspond to the service name, and the field. Statements correspond to the ROS service.

The code snippet entity extraction method shown in Figure 4 primarily focuses on extracting entities related to the 'subscribe' topic and the 'advertise' service. In this case, if the function field refers to 'subscribe', the value/arguments correspond to the topic name, but the ROS message cannot be found in the code snippet. To extract the ROS message corresponding to the topic, we need to extract the code related to the ROS node and find the ROS message associated with the called topic. Similarly, if the function/field refers to 'advertise Service', the value/arguments correspond to the service name. Similarly, the ROS service cannot be extracted from this code snippet. To extract the ROS service corresponding to the service, we need to extract the code related to the ROS node and find the ROS service associated with the called service.

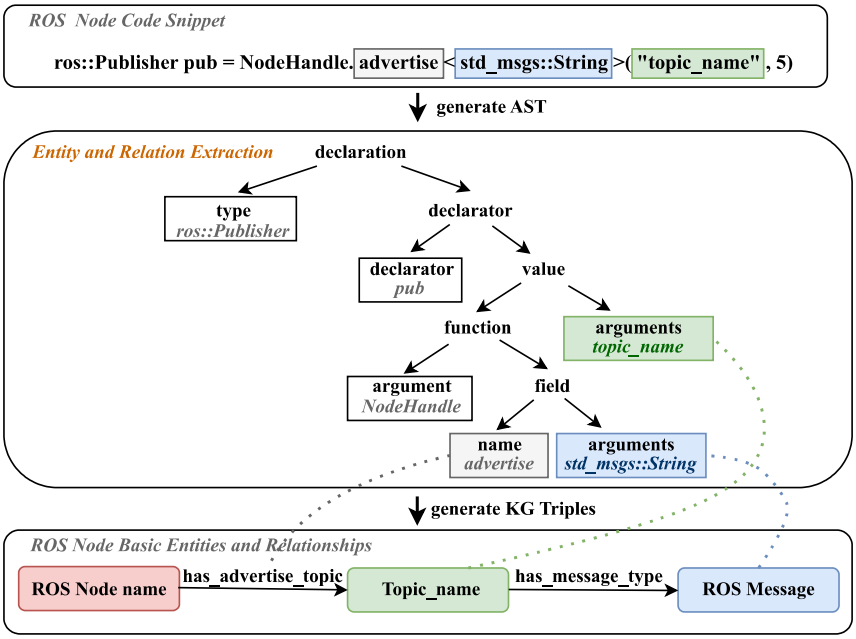

Figure 3. Method for extracting topic entities, message entities, and their corresponding relationships from code snippets (advertise).

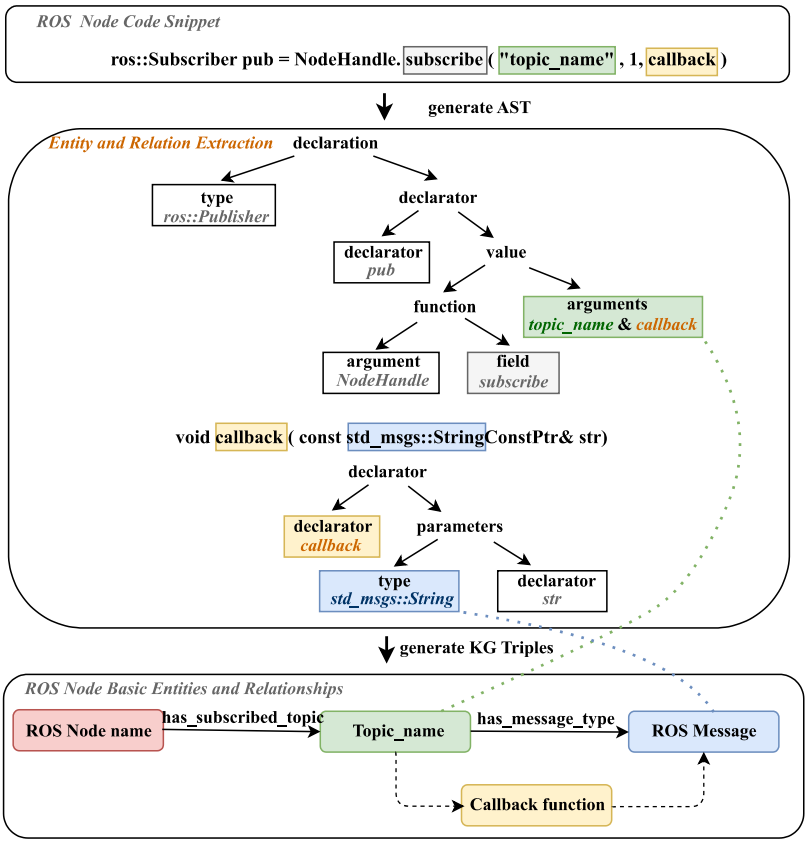

Figure 4. Method for extracting topic entities, message entities, and their corresponding relationships from code snippets (subscribe).

3.2.3. Functionality Classification

The functional description of ROS nodes refers to the specific tasks and functionalities that nodes perform. However, obtaining these functional descriptions of ROS nodes poses significant challenges. Firstly, the description information about ROS nodes may be scat-

tered across various source code and documentation fragments. Finding a comprehensive and detailed functional description requires extensive searching and reading, incurring high costs in terms of acquisition. Secondly, these functional descriptions of ROS nodes are often authored by different developers, resulting in varying quality levels and making it more difficult for developers to understand and utilize the nodes. Therefore, obtaining a thorough and explicit description of ROS nodes is highly challenging.

However, ROS nodes are typically named following certain conventions that often reflect their intended functionalities and tasks. Therefore, reasonable speculations based on the names of ROS nodes can help us gain insights into their functional domains. Furthermore, within the knowledge graph of ROS nodes, it is necessary to incorporate functional entities related to the names of ROS nodes.

Considering the diverse functionalities and modules of robot systems, we can categorize ROS nodes into the following groups: hardware nodes, data processing nodes, control nodes, navigation nodes, human–robot interaction nodes, analysis and detection nodes, and simulation nodes.

- Hardware nodes are responsible for interacting with the physical components of the robot, such as sensors, actuators, and other hardware device;
- Data processing nodes perform various data processing tasks, including filtering, feature extraction, transformation, and fusion, to extract meaningful information from raw sensor data;
- Control nodes execute algorithms and logic to regulate the robot's behavior and achieve desired actions based on sensor input and system state;
- Navigation nodes handle tasks related to robot motion planning, localization, mapping , and obstacle avoidance, and enable the robot to navigation autonomously in its environment;
- human–robot interaction nodes facilitate interaction between the robot and humans, encompassing speech recognition, gesture control, graphical user interface, and other modalities for seamless human–robot interaction;
- Analysis and detection nodes focus on analyzing and detecting specific patterns, objects, or events in sensory data, which can support higher-level perception and decision-making processes;
- Simulation nodes provide simulated environments and virtual models to test and evaluate robot behaviors, algorithms, and system performance in a controlled and reproducible manner.

These entities provide a rough description of the functionality of ROS nodes and, when combined with the existing entities obtained from the code, help developers quickly understand and select appropriate ROS nodes, reducing the cost of searching and reading to enhance development efficiency.

3.2.4. Hardware Classification

Storing the hardware dependencies of ROS nodes as entities in the ROS node knowledge graph enables a quick understanding of the hardware requirements for each node. This facilitates the sharing and reuse of relevant ROS node knowledge and experiences. By including hardware entities in the knowledge graph, developers and researchers can easily access information about the specific hardware components needed by each ROS node.

By performing part-of-speech (POS) tagging on the textual descriptions of ROS nodes and using regular expressions or other text processing techniques, we can extract noun phrases that adhere to specific tagging patterns and filter out hardware-related information. Firstly, we annotate the textual descriptions of ROS nodes with their corresponding POS tags. Based on the specific domain of ROS nodes and the corpus being used, we created a set of regular expressions that match and extract hardware-related information from the ROS node descriptions, as shown in Table 4. Eventually, we used constructed regular expressions for POS-tagged ROS node descriptions, searching for combinations of words that fit the hardware-related patterns.

Table 4. The extraction of hardware information using regular expressions.

Regular Expressions
<NNS>*<NNP>*<CD>, <NNP>*, <JJ>+<NN>+, <JJ>+<NNP>+ <NNP>*<NN><NNP>, <NNP>+<NNS>*<NN>*, <NNS>*<NN>+

Although we successfully extracted the hardware information from ROS nodes, it remains a domain-specific field that can be challenging for developers to comprehend. In order to facilitate a more user-friendly understanding and utilization of ROS nodes, we propose a further detailed categorization of the hardware attributes associated with ROS nodes in Table 5. By breaking down the hardware attributes into more granular subcategories, we were able to systematically capture and organize information such as hardware types and compatibility considerations.

Table 5. A classification of ROS node hardware.

Category	Subcategory
Robots	Aerial, Component, Ground, Manipulator, Marine
Sensors	1D range finders, 2D range finders, 3D Sensors, Audio device, Cameras, Enviromental, Force Sensors, Motion Capture, Pose Estimation, Power Supply, RFID, IO Interfaces, Speed
Motor	Motor Controller, Servo Controller

By employing the aforementioned approaches, we were able to successfully extract the functional and hardware information of ROS nodes from unstructured data. These techniques enabled us to transform the textual descriptions and dependencies of ROS nodes into structured and categorized representations.

3.3. ROS Node Knowledge Graph Construction

Through the aforementioned methods and steps, we successfully constructed a knowledge graph of ROS nodes. Leveraging natural language processing and code analysis techniques, we were able to extract entities such as nodes, topics, services, and messages from ROS code and related information, organizing them in a structured manner within the knowledge graph. Tables 6 and 7 present the entities and relationships within the ROS node knowledge graph.

Table 6. Entities of the ROS node knowledge graph.

Entity Name	Entity Description
ROS nodes	ROS nodes are independent units of execution that perform specific functionalities or algorithms. They communicate with each other through the publication and subscription of topics and by calling and providing services.
Topics	Topics serve as channels for message exchange between ROS nodes. A node can publish messages to a topic, and other nodes can subscribe to that topic to receive the messages.
Message types	Message types define the data structure for communication between ROS nodes.
Services	Services enable communication between ROS nodes in a request–response fashion. One node can provide a service, and another node can call that service to request a specific functionality.
Service types	Service types define the data structure used for communication between ROS nodes during service calls.

Table 6. *Cont.*

Entity Name	Entity Description
ROS package	An ROS package is the fundamental organizational unit for managing ROS code.
Code file URLs	Uniform resource locators (URLs) pointing to the code files of ROS nodes.
Function	Function entities represent specific functionalities or algorithms provided by nodes.
Hardware	Hardware entities refer to physical devices, such as sensors, actuators, etc., that interact with ROS nodes.

Table 7. Relationships of the ROS node knowledge graph.

Relationship	Relation Description
has_published/ subscribed_topic	describes the connection between an ROS node and the topics it publishes to or subscribes to.
has_provided/ called_service	describes the connection between an ROS node and the services it calls or provides.
has_message_type	describes the association between a topic and its message type.
has_service_type	describes the association between a service and its service type.
has_package	describes the association between an ROS node and the ROS package it belongs to.
has_code_url	describes the association between an ROS node and the URL of its code file.
has_function	describes the association between an ROS node and a function entity.
has_hardware	describes the association between an ROS node and a hardware entity.

The schema of the knowledge graph is displayed in Figure 5. The constructed ROS node knowledge graph consists of 12,350 entities and 36,029 triples. Considering that some ROS nodes may not have inputs and outputs with regard to services, each ROS node is connected to 5.17 (<6) entities. Additionally, due to the inclusion of hardware and function knowledge using entity typing, the average path length from an ROS node to hardware and function is 2.97 and 2, respectively. In the Figure 6, the ROS node 'keyboard_cmd' is depicted.

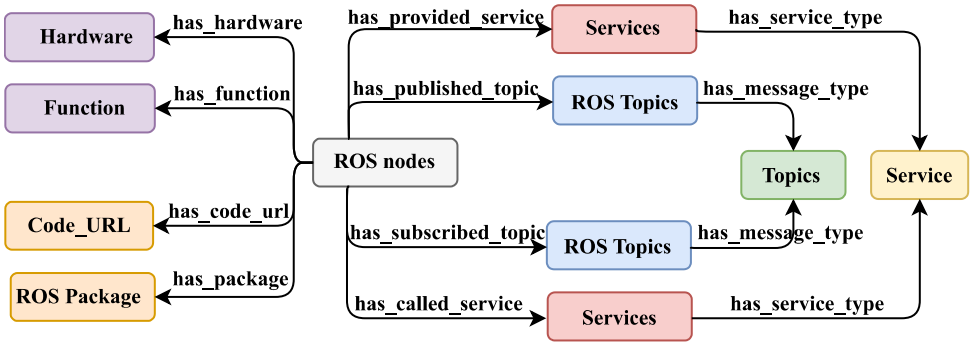

Figure 5. The core ontology of ROS node knowledge graph.

This knowledge graph provides researchers and developers with a comprehensive and accurate resource to gain a deeper understanding of the structure, functionality, and interrelationships of ROS nodes. With this graph, robotics developers can explore the topics, services, and messages involved in specific nodes, facilitating a better analysis and comprehension of the operational mechanisms of ROS systems.

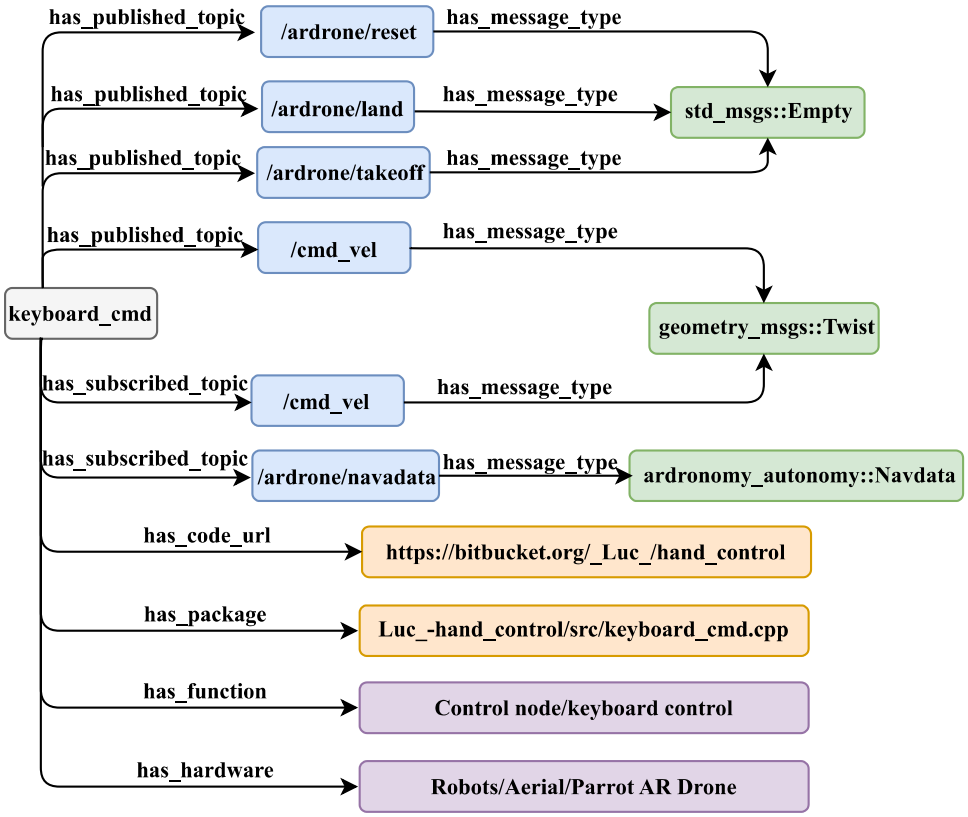

Figure 6. An example of ROS node 'keyboard_cmd'.

4. Evaluation

In order to validate the quality of the constructed ROS node knowledge graph (RNKG) and determine its utility and reliability in providing node information, node relationships, and node search capabilities, a series of experiments were conducted. Firstly, the completeness of the RNKG was assessed by examining the presence of comprehensive node information within the knowledge graph. Secondly, the accuracy of the edges in the RNKG was evaluated to ensure the correctness of the relationships between various entities in the ROS node knowledge graph. Lastly, the constructed RNKG was tested against user search requirements to identify ROS nodes that fulfill their needs, thereby validating its suitability for developers' search demands. These experiments were instrumental in understanding whether the constructed RNKG can meet the requirements of ROS developers and assess its potential in enhancing the development efficiency of robotic systems.

4.1. The Completeness of RNKG

The dataset required to build the knowledge graph was collected in open-source repositories. It needs to be judged whether the RNKG contains all needed ROS node information, including ROS nodes, topic, service, and message.

4.1.1. Protocol

To ascertain the completeness of the knowledge graph for ROS nodes, it was necessary to evaluate it from various perspectives.

Firstly, we verified whether the ROS node knowledge from the ROS Wiki was present in the knowledge graph for ROS nodes. Since the ROS Wiki serves as the official resource repository for ROS, it contains information about ROS nodes. By utilizing the relevant documentation provided in the ROS Wiki, a comparison was made between the node information in the knowledge graph for ROS nodes and the constructed ROS node knowledge graph. This comparison helped determine whether the ROS node knowledge from the ROS Wiki was fully incorporated within the knowledge graph, thus validating its completeness.

Secondly, a subset of ROS node knowledge was selected from the constructed knowledge graph, and its representation in the actual code was examined to ensure the accurate manifestation of the corresponding entities. This step allowed us to ascertain the reliability and completeness of the ROS node knowledge graph concerning the ROS node knowledge.

By employing these evaluation methods, we were able to rigorously assess the completeness of the knowledge graph for ROS nodes, ensuring that it comprehensively captured relevant information from the official ROS Wiki and accurately reflected the corresponding entities and their relationships in the implemented code.

4.1.2. Results and Analysis

Table 8 presents the completeness comparison results between the RNKG (ROS node knowledge graph) and ROS Wiki. It indicates that the majority of ROS node knowledge from the ROS Wiki could be found in the RNKG. Therefore, it can be said that the idea of extracting ROS node knowledge from open-source projects to build the ROS node knowledge graph is feasible and, to some extent, provides evidence that the ROS node knowledge in our constructed graph is complete.

Table 8. The completeness of RNKG compared with ROS Wiki.

Category	ROS Nodes	Topic	Service	Message
From ROS Wiki/RNKG	345/340	345/332	345/317	345/308
Contained Percentage	98.55%	96.23%	91.88%	89.27%

Table 9 displays the matching results between the knowledge in the ROS node knowledge graph (RNKG) and the knowledge found in the code files. The data indicate that the vast majority of ROS node knowledge from the code files could be found in the RNKG. This finding highlights the completeness of the RNKG in accurately capturing and representing the ROS node knowledge present in the actual code implementation.

Table 9. The completeness of RNKG.

Category	Precision	Recall	F1
Topics (subscribed)	87.50%	87.72%	87.60%
Topics (published)	87.93%	86.44%	87.17%
Services (provided)	95.71%	97.14%	96.14%
Services (needed)	89.71%	86.97%	88.31%
Service types	93.63%	88.55%	91.01%
Message types	92.07%	94.37%	93.20%
ROS package	93.45%	91.74%	92.58%

Overall, the established completeness of the ROS node knowledge graph highlights its value as a comprehensive and reliable resource, supporting ROS developers throughout the software development life cycle and enabling them to leverage the full potential of the ROS ecosystem.

4.2. The Correctness of ROS Node Knowledge in RNKG

To ensure the usability of the ROS node knowledge graph for supporting robot software development, it was crucial to assess the correctness of relationships between entities within the ROS node knowledge graph.

4.2.1. Protocol

Validating the correctness of relationships within the ROS node knowledge graph was essential, as the interactions of ROS nodes with topics, services, and messages are crucial. To verify the accuracy of the knowledge graph, it was, indeed, valuable to manually assess the relationships between different entities. Randomly selecting a series of nodes that had existing relationships and evaluating whether their associations with related entities were correct was a valid approach.

We randomly selected a set of nodes from the ROS node knowledge graph that had known relationships with topics, services, or messages. For each selected node, we verified whether it was correctly linked to the relevant entities, such as topics, services, or messages. In addition, we ensured that the associations accurately represented the actual interactions.

4.2.2. Result

The data presented in Table 10 represent the results of assessing the correctness of relationships among different entities in the knowledge graph. From the data, it is evident that the recall of the ROS node knowledge graph we constructed is relatively high. This indicates that the knowledge graph accurately captured the relationships between many entities, thereby reflecting the actual interactions among ROS nodes.

Table 10. The correctness of relationship in RNKG.

Triplets in RNKG	Precision	Recall	F1
(Topics, has_subscribed_topic ,Message types)	69.55%	81.72%	75.14%
(Topics, has_published_topic ,Message types)	87.72%	86.44%	87.07%
(Services, has_provided_service, Service types)	83.80%	87.14%	85.43%
(Services, has_called_service, Message types)	74.52%	86.97%	80.26%

Consequently, the constructed knowledge graph effectively represents the relationships between ROS nodes and provides a reliable reference resource for robot software developers.

4.3. The Effectiveness of RNKG in Searching ROS Nodes

The primary objective of constructing the ROS node knowledge graph was to facilitate the development of robot software for ROS developers. The main purpose of designing this experiment was to assist developers in quickly searching the desired ROS nodes.

4.3.1. Protocol

To validate that the constructed ROS node knowledge graph can assist developers in searching for ROS nodes, we compared the ROS node search algorithm based on the ROS node knowledge graph with "Keyword Search", "Fuzzy Search", and "Text Search". Additionally, to measure the effectiveness of the ROS node search method, we primarily focused on the normalized discounted cumulative gain (NDCG) metric [16].

- Keyword search: TF-IDF (term frequency-inverse document frequency) [17] is a statistical method used to evaluate the importance of words in a text, and it is widely applied in the field of information retrieval. By using the TfidfVectorizer from the scikit-learn library, we were able to perform feature extraction and vectorization of the ROS nodes corpus;
- Fuzzy search: While ROS documentation is not specifically designed as a website to search for ROS nodes, it is possible to find the desired ROS nodes by performing fuzzy matching on a large number of ROS document name. Despite its limitations in searching for ROS nodes, this method can be considered an effective approach provided by the official ROS resources;
- Text search: SentenceBERT [18] is a deep learning model specifically designed to handle the problem of embedding natural language text. In the ROS node search task, it is possible to calculate the similarity between user queries and ROS node description sentences.

This was done to confirm that building the ROS node knowledge graph does, indeed, help developers find ROS nodes more effectively.

4.3.2. Result

The performance of the ROS node search based on the ROS node knowledge graph can be observed in Table 11. From the data presented in the table, it is evident that the ROS

node search method based on the ROS node knowledge graph outperformed other ROS node search methods in terms of accuracy when searching for ROS nodes. The accuracy improvement across all methods was more than 10%, indicating a significant enhancement in assisting the task of locating ROS nodes using the ROS node knowledge graph we constructed. This confirms the utility of the ROS node knowledge graph we built in aiding ROS developers in robot software development.

Table 11. Comparison of different ROS node search methods.

Method Classification	Method	NDCG@1		NDCG@5	
		Accuracy	Performances	Accuracy	Performances
Keyword-Based	TF-IDF	0.3377	−59.69%	0.5746	−43.37%
Subgraph Matching	VF2++	0.2575	−66.15%	0.4112	−47.62%
Fuzzy Search	ROS Doc	0.4562	−38.23%	0.6738	−14.17%
NLP Search	SentenceBert	0.5458	−26.10%	0.68077	−13.28%
-	RNKG-based	0.7386		0.7851	

The ROS node knowledge graph is crucial for ROS developers during robot software development. It provides a structured and reliable approach for acquiring, learning, and referencing ROS node information, bolstering development efficiency and accuracy, and fostering rapid advancement in robot software.

5. Conclusions and Future Work

In this paper, we propose a method to construct a robotic node knowledge graph (RNKG) with 12,350 entities and 36,029 triples for better serving ROS developers. We systematically describe how it was automatically built from source code knowledge from open-source robotics repositories and ROS Wiki. Following the previous two experiments, which demonstrated the completeness and accuracy of the constructed knowledge graph, we conducted another experiment to validate the effectiveness of the knowledge graph in assisting robot development. The experiment ultimately confirmed that the constructed ROS node knowledge graph is capable of supporting robot development and provides accurate representations of ROS nodes through structured knowledge.

In the future, we plan to continue in-depth research on the ROS node knowledge graph from two perspectives. Firstly, we aim to explore how to integrate code knowledge that is highly relevant to functionality into the ROS node knowledge graph. Secondly, we want to investigate how to leverage the ROS node knowledge graph to accelerate the development process of robot software. After in-depth mining of robot software project information, the new data acquired by it will be integrated into the constructed RNKG. We will continually enlarge the RNKG to cover emerging ROS nodes. On the other hand, we can better apply the reasoning ability of the knowledge map to the development tasks of robot software and use better semantic technology to support developers in searching for the ROS node they need.

Author Contributions: Conceptualization, Supervision and funding acquisition, X.M.; methodology, software, validation, writing—original draft preparation, Y.Z.; software and validation, writing—review and editing Y.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Key Laboratory of Software Engineering for Complex Systems and the National Science Foundation of China under granted number 62172426. This research was supported by the Scientific Research Project of Education Department of Hunan Province, China, under Grant 18A470.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A.Y. ROS: An open-source Robot Operating System. In Proceedings of the ICRA Workshop on Open Source Software, Kobe, Japan, 12–17 May 2009; Volume 3, p. 5.
2. Estefo, P.; Simmonds, J.; Robbes, R.; Fabry, J. The robot operating system: Package reuse and community dynamics. *J. Syst. Softw.* **2019**, *151*, 226–242. [CrossRef]
3. Xu, J.; He, M.; Jiang, Y. A novel framework of knowledge transfer system for construction projects based on knowledge graph and transfer learning. *Expert Syst. Appl.* **2022**, *199*, 116964. [CrossRef]
4. Ore, J.P.; Elbaum, S.; Detweiler, C. Dimensional inconsistencies in code and ROS messages: A study of 5.9 M lines of code. In Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017; pp. 712–718.
5. Kolak, S.; Afzal, A.; Le Goues, C.; Hilton, M.; Timperley, C.S. It takes a village to build a robot: An empirical study of the ROS ecosystem. In Proceedings of the 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), Adelaide, Australia, 28 September–2 October 2020; pp. 430–440.
6. Witte, T.; Tichy, M. Inferred interactive controls through provenance tracking of ros message data. In Proceedings of the 2021 IEEE/ACM 3rd International Workshop on Robotics Software Engineering (RoSE), Madrid, Spain, 2 June 2021; pp. 67–74.
7. Bédard, C.; Lajoie, P.Y.; Beltrame, G.; Dagenais, M. Message flow analysis with complex causal links for distributed ROS 2 systems. *Robot. Auton. Syst.* **2023**, *161*, 104361. [CrossRef]
8. Chen, L.; Mao, X.; Zhang, Y.; Yang, S.; Wang, S. An efficient ros package searching approach powered by knowledge graph. In Proceedings of the International Conference on Software Engineering and Knowledge Engineering, Virtual Conference, 1–10 July 2021; pp. 411–416.
9. Hart, S.; Dinh, P.; Hambuchen, K. The affordance template ROS package for robot task programming. In Proceedings of the 2015 IEEE international conference on robotics and automation (ICRA), Seattle, WA, USA, 26–30 May 2015; pp. 6227–6234.
10. Bozhinoski, D.; Aguado, E.; Oviedo, M.G.; Hernandez, C.; Sanz, R.; Wąsowski, A. A Modeling Tool for Reconfigurable Skills in ROS. In Proceedings of the 2021 IEEE/ACM 3rd International Workshop on Robotics Software Engineering (RoSE), Virtual Conference, 22–30 May 2021; pp. 25–28.
11. Timperley, C.S.; Dürschmid, T.; Schmerl, B.; Garlan, D.; Le Goues, C. Rosdiscover: Statically detecting run-time architecture misconfigurations in robotics systems. In Proceedings of the 2022 IEEE 19th International Conference on Software Architecture (ICSA), Honolulu, HI, USA, 12–15 March 2022; pp. 112–123.
12. Bo, S.; Mao, X.; Yang, S.; Chen, L. Towards An Efficient Searching Approach of ROS Message by Knowledge Graph. In Proceedings of the 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC), Los Alamitos, CA, USA, 27 June–1 July 2022; pp. 934–943.
13. Bouziane, R.; Terrissa, L.S.; Ayad, S. Semantic web services for ROS: A Robot as a Service approach. *Autom. Softw. Eng.* **2022**, *29*, 49. [CrossRef]
14. Daruna, A.; Gupta, M.; Sridharan, M.; Chernova, S. Continual learning of knowledge graph embeddings. *IEEE Robot. Autom. Lett.* **2021**, *6*, 1128–1135. [CrossRef]
15. Malavolta, I.; Lewis, G.; Schmerl, B.; Lago, P.; Garlan, D. How do you architect your robots? State of the practice and guidelines for ROS-based systems. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice, Seoul, Republic of Korea, 27 June–19 July 2020; pp. 31–40.
16. Wang, Y.; Wang, L.; Li, Y.; He, D.; Liu, T.Y. A theoretical analysis of NDCG type ranking measures. In Proceedings of the Conference on Learning Theory, Princeton, NJ, USA, 12–14 June 2013; pp. 25–54.
17. Martineau, J.; Finin, T. Delta tfidf: An improved feature space for sentiment analysis. In Proceedings of the International AAAI Conference on Web and Social Media, San Jose, CA, USA, 17–20 May 2009; Volume 3, pp. 258–261.
18. Reimers, N.; Gurevych, I. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv* **2019**, arXiv:1908.10084.