API for an Autonomous Mobile Service Robot

Raphaël de Araújo Colcombet
raphael.colcombet@ist.utl.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2023

Abstract

This thesis explores the fundamental structures and results of the SocRob@Home API for an Autonomous Mobile Service Robot, highlighting the effectiveness of general-purpose programming languages in establishing a useful abstraction layer. The main purpose of this abstraction is to streamline robot configuration, operation, and control by abstracting away complex technological details, providing a more user-friendly interface for the development of future robotic applications. The TiagoAPI, central to this project, offers an object-oriented approach designed for seamless integration with a wide range of robotic platforms. It simplifies intricate robotic processes into comprehensible commands, enabling efficient and versatile robot operation. The API is rooted in modern software paradigms and offers a comprehensive suite of features such as advanced navigation, object handling, perceptual capabilities, linguistic processing, and strategic task formulation. This work underscores the need for user-centric robotic software frameworks. By rendering intricate robotic components more accessible through the TiagoAPI, the research contributes towards bridging the gap between advanced robotic capabilities and user accessibility, targeting a diverse user base with varied technical proficiency.

Keywords: Robotic architecture, Robot API, API, ROS;

1. Introduction

Robotics has emerged as a valuable tool across diverse sectors such as industry [2], science [21], the military [25], health [15], and education [3]. This multidisciplinary field necessitates expertise in a blend of disciplines like engineering, computer science, and materials science for its development [30].

In the era of the information revolution, characterized by rapid digital technology advancements and widespread adoption, robotics has evolved in tandem, playing a crucial role in various industries and holding the potential to transform people's lifestyles, work dynamics, and interactions with technology [4].

Within the broader realm of robotics, the segment of domestic robots is rapidly expanding [8]. These robots are tailored for an array of domestic tasks like cleaning, cooking, and providing companionship [23]. With continuous enhancements in capabilities and decreasing costs [7], domestic robots are gaining popularity globally.

The SocRob@Home project [24], an initiative of the Institute for Systems and Robotics (ISR) research group at Instituto Superior Técnico (IST) since 1998, has been a significant contributor to this progress. It has a remarkable history of accomplishments in prestigious competitions like RoboCup 2018, ERL Smart Cities 2019, and RoboCup 2021, with its recent commendable performance securing second place at RoboCup 2023 in Bordeaux, as depicted in Figure 1. This achievement signifies a substantial advancement, demonstrating the team's dedication to the advancement of robotics. The success not only reflects the efforts of the current team members but also acknowledges the contributions of former students involved in the project.

Figure 1: RoboCup 2023 Award Ceremony - SocRob@Home

1.1. Context

An Application Program Interface (API) is a set of guidelines and standards for software access [22]. A robot API enables software programs to communicate and share data [11]. It's crucial in technology, where multiple systems collaborate for complex tasks [36]. For example, it lets a robot access sensor data for navigation or communicate with users through natural language processing [28]. This empowers developers to easily integrate diverse functionality into systems, making them more advanced and flexible [13]. A significant part of robotic functionalities relies on ROS, which, when accessed without an API isn't user-friendly.

Additionally, a robot API offers cost savings, standardizes interfaces, and abstracts underlying functions [26] [20]. This abstraction allows developers to focus on high-level goals, rather than intricacies of components [35].

For autonomous mobile service robots, implementing an API poses challenges due to complex sensor and software integration, but it also opens avenues for innovative applications [35]. An intuitive API facilitates development, enhancing robot usefulness [35].

Even non-developers can employ a robot API to interact with Robot Operating System (ROS) components without deep software knowledge. This not only organizes software but promotes modularization and compartmentalization, facilitating functionality transfer between robot systems. However, creating a universally applicable and reliable robot API remains a challenge, given the evolving nature of robotics. Developers must stay updated and adapt proactively.

1.2. Motivation

This thesis seeks to develop a programming interface for the TIAGo mobile service robot, incorporating diverse functionalities like navigation, manipulation, grasping, and speech understanding. The objective is to create an accessible API that simplifies the invocation of these functions for programmers, enabling them to seamlessly combine them in various tasks. The key steps involve comprehensive classification of the implemented functions and the eventual open-sourcing of the API for community use and expansion. Additionally, the project aims to enhance understanding of different robot subsystems and expertise in integrating robot systems.

1.3. Document Structure

The structure of this document is organized to provide a thorough understanding of the research and methodologies employed in the thesis entitled: "API for an Autonomous Mobile Service Robot."

It is structured as follows:

- Background: Introduces foundational tools and platforms for the project, highlighting Python, ROS, and the TIAGo robot's functionalities.

- Related Work: Reviews key research in robotic technologies with a focus on robot API. Insights are drawn from works such as the Robotics API, YARP, Microsoft's RDS, and OROCOS, showcasing the fusion of robotic techniques with contemporary programming paradigms.

- Proposed Solution: Details TiagoAPI, highlighting the resource-oriented API, its architecture, and essential features like Auto Diagnostics Service and the API CMD Tool.

- Experiments and Results: Provides insights into the TiagoAPI's performance in simulated and real-world environments affirming its potential for diverse robotic applications.

- Conclusions: Presents concluding observations and charts potential avenues for enhancing and broadening the API's scope in future research.

2. Background

This section delves into the key components underpinning the SocRob@Home 2023 initiative. It scrutinizes the preference for Python as the principal programming language, the diverse functionalities of the ROS platform, and the distinctive attributes of the TIAGo robot. These carefully selected tools, valued for their adaptability and pertinence, serve as the core of our domestic robotics advancements.

2.1. Python

Python, created by Guido van Rossum in 1991, is a popular programming language for its flexibility, user-friendliness, and extensive community [37]. With its rich libraries and interpreted nature, Python is highly adaptable across various platforms, allowing for rapid development and deployment [32, 18].

ROS supports Python and C++, each with distinct strengths and weaknesses [33]. While C++ offers tight integration with ROS due to its use in the core libraries, Python's ease of use and strong community support make it a viable choice [19].

Choosing Python for the project was driven by the need for seamless integration with existing systems, facilitating rapid iterations, and modifications. Since the project's API didn't require intensive computations, Python's simplicity and readability helped streamline development for the team's new members.

2.2. ROS

ROS is a popular framework for constructing and managing robotic systems, introduced by Willow Garage in 2007 [5]. It facilitates the development of distributed systems and implementation of common robot functionalities. ROS comprises four core components:

- Messaging: ROS provides a robust publish-subscribe communication system, enabling efficient interaction between different robot components [5]. This mechanism ensures reliable data exchange and supports modular design through Inter-Process Communication (IPC).
- Utilities: ROS offers diverse utilities for configuration management, debugging, visualization, and system introspection, simplifying the deployment and maintenance of robotic systems [33].
- Libraries: It encompasses a comprehensive collection of libraries for essential robot functionalities, including mobility, manipulation, and perception [5].
- Ecosystem: Backed by an active community, the ROS ecosystem provides access to extensive knowledge, collaborative resources, and a repository of ROS packages for seamless integration of third-party components [33].

ROS finds applications in diverse fields such as mobile robotics, aerospace, manufacturing, and service robotics [5]. It offers a unified set of APIs for common tasks, reducing development time and effort [33]. However, the reliance on a publish-subscribe messaging model can lead to increased overhead and latency, and integrating additional libraries may introduce complexity, demanding a learning curve for newcomers [5, 33]. Despite these considerations, ROS remains a powerful choice for building and managing robotic systems, offering compatibility, a rich library of tools, and a supportive community [27].

2.3. TIAGo Robot

The SocRob@Home 2023 project utilizes the TIAGo mobile service robot from PAL Robotics (see Fig. 2), renowned for its adaptability in addressing diverse challenges of domestic robotics.

TIAGo, a mobile manipulator robot, mimics human-like body and limbs, facilitating intricate interactions with the environment. Equipped with a comprehensive sensor suite, robust processing unit, and sophisticated software, it is well-suited for the project's diverse research domains.

Key attributes of the TIAGo Steel robot include a differential-drive base, a front laser range-finder, and rear sonars, enabling effective navigation and obstacle avoidance in domestic environments. Its 7-degree-of-freedom arm, parallel gripper, and lifting torso facilitate tasks like object manipulation and pick and place operations. Moreover, its speaker and stereo microphone enhance human-robot interaction.

Figure 2: PAL Robotics TIAGo Steel Edition Robot

Equipped with an RGB-D camera, TIAGo facilitates object detection, people tracking, and obstacle perception. Its adaptability and configurability make it invaluable for the project's evolving research needs, aligning perfectly with objectives in ambient-assisted living, healthcare, and light industry.

In summary, the TIAGo mobile service robot is central to the SocRob@Home 2023 project, effectively tackling the complex challenges of domestic robotics through its humanoid-inspired design, inclusive sensors, and strong functionalities. It excels in mapping, navigation, perception, manipulation, decision-making, and human-robot interaction, making it indispensable.

3. Related Work

In the study of robotic technologies, acknowledging past research is crucial for driving progress and bridging complexities. This section delves into relevant research on robot APIs, particularly those surpassing hardware limitations. By examining studies in various scenarios, we aim to elucidate techniques, challenges, and outcomes. This analysis not only enhances understanding of prior work but also underscores the distinctiveness and benefits of our simplified and adaptable approach, detaching software complexities from hardware constraints. Through this perspective, readers can grasp the context and advancements in enhancing robotic control accessibility and adaptability.

3.1. Robotics API Overview

In [1], Angerer introduces the Robotics API, revolutionizing industrial robot programming by embracing an object-oriented approach, contrasting with traditional, restrictive languages like ALGOL or

Pascal. The API's robustness is apparent in its 70 classes, facilitating the modeling of real-world entities and enhancing code reusability.

A distinctive attribute is its command handling, drawing inspiration from the "Command pattern" outlined in [10], enabling seamless integration of multiple commands with precise timing.

Leveraging Microsoft's C# and .NET framework, the API prevents common programming errors, ensuring efficient development, exemplified by the flawless execution of a welding task.

The API's practicality is demonstrated through its application in controlling KUKA lightweight robots, with future plans focusing on expanded sensor integration and refined error handling. It serves as a groundbreaking innovation, simplifying advanced robotic functionalities, particularly for smaller businesses.

Overall, the Robotics API symbolizes the union of conventional robotics and contemporary programming paradigms, serving as a source of inspiration for the development of TiagoAPI.

3.2. YARP: A Flexible and Adaptable Robot Platform

In the realm of robotic software frameworks, YARP (Yet Another Robot Platform) stands out for its ability to seamlessly connect diverse robot parts and computational devices [29]. Unlike proprietary systems, YARP is open-source and modular, promoting a universal development approach through its abstraction of inter-device communication complexities. This is achieved via the creation of ports that enable bidirectional data exchange across different machines [29].

YARP's strength lies in its simplicity and adaptability, offering fundamental network communication primitives that developers can customize to meet their specific needs [9]. Its object-oriented design enables easy encapsulation of real-world robotic components, sensors, and actuators.

The platform's support for middleware allows integration of various communication protocols, or custom solutions, without necessitating core logic alterations [29]. YARP has proven its utility in controlling diverse robotic platforms, ranging from humanoid robots to manipulative arms, underscoring its potential for bridging hardware gaps and real-time constraints [31].

YARP's recognition and practical application led to the development of the TiagoAPI, which follows YARP's principles to cater to specific developer needs, ensuring reliability and adaptability [29].

3.3. Microsoft's RDS: Robotics Developer Studio

Microsoft's Robotics Developer Studio (RDS), simplifies complex robotic application development [17]. It employs a modular, object-oriented approach, enhancing reusability and organization [6].

A prominent feature is Visual Programming Language (VPL), offering a user-friendly graphical interface for designing and monitoring robotic applications, promoting quick prototyping [17].

Apart from VPL, RDS boasts decentralized services for sensor interfacing, motion planning, and diagnostics, aligning with object-oriented principles [16].

Notably, Concurrency and Coordination Runtime (CCR) addresses concurrent operations, ensuring efficient multitasking with multi-core processors [17].

The included simulator enables risk-free application testing, accelerating development and instilling confidence in solutions [6].

By leveraging Microsoft's software products and .NET framework, RDS unifies diverse tools, libraries, and protocols, making it accessible for all [16].

Overall, RDS set the precedent for TiagoAPI's comprehensive and versatile robotic APIs through its pioneering approach and industry-leading practices.

3.4. OROCOS: Open Robot Control Software

OROCOS, an influential open-source framework for robot control, is renowned for its emphasis on modularity and real-time operation [34]. Written in C++, it offers a range of advanced control libraries and has evolved into an extensive ecosystem for robotic software development, embracing various middleware and tooling avenues [14].

With the Real-time Toolkit (RTT) and the Kinematics and Dynamics Library (KDL), OROCOS enables timely and adaptable robot operations, showcasing the importance of intuitive and flexible platforms for developers [38]. Its modular and component-based architecture embodies the concept of viewing functionalities as distinct resources, which could inform the development strategy of the TiagoAPI project [38].

OROCOS's integration emphasis, particularly with the ROS platform, offers valuable insights on ensuring transferability and compatibility, which can enhance TiagoAPI's integration with existing ROS components [12]. By simplifying robotic interactions, OROCOS has set a benchmark for broader developer accessibility, a guiding principle that could inspire the evolution of TiagoAPI into a robust and versatile robotic interface [14].

4. Proposed Solution

This thesis introduces a solution for crafting an interface to define TIAGo's functions and algorithms. The key objective is to offer a user-friendly and adaptable interface that empowers developers to delineate TIAGo's capabilities and regulate its opera-

tions. Our approach entails developing a resource-oriented API that uniformly exposes robotic resources, streamlining the interaction with the system for developers of varying proficiencies and backgrounds. Subsequent sections will delve into the specifics of our devised solution.

4.1. TiagoAPI

To simplify robot control without relying on ROS, we created a user-friendly Python API named TiagoAPI. This API lowers the barriers for users, enhancing accessibility. An illustrative example from the API documentation is provided in Listing 1.

Listing 1: Pseudocode for using TiagoAPI

```
INITIALIZE TiagoAPI as tiago_api
FUNCTION greeting(location):
    CALL tiago_api.navigation.moveTo
        with a location as GOAL
    CALL tiago_api.manipulation.
        runPredefinedMotion with "
        wave" as MOTIONNAME
    CALL tiago_api.hri.sayAndWait
        with "Hello, World!" as TEXT
END FUNCTION
```

This approach enables seamless integration with existing ROS components, eliminating the need to manually configure ROS subscribers or service clients for communication. The robot API handles these tasks discreetly, without requiring user intervention. Additionally, it can initialize a ROS node using `rospy.init_node()` when necessary, creating a new node if none is active within the current process.

For the development of TiagoAPI, we adopted the singleton design pattern. This choice holds particular significance as it ensures centralized and consistent control over the robot's resources and state. Such centralized control improves performance, predictability, and reliability. The singleton pattern also provides a global access point to the single class instance, simplifying interactions with the robot's resources for other parts of the system. This streamlined control is crucial for achieving synchronized and efficient robotic operation.

This choice guarantees a sole instance of the `TiagoAPI` class. Encapsulated within is the `__TiagoAPIInstance` class, embodying the actual implementation of the API's features. Figure 3 displays the Unified Modeling Language (UML) class diagram of the TiagoAPI.

The relationship between `TiagoAPI` and `__TiagoAPIInstance` is compositional. In the context of the Singleton pattern, the `TiagoAPI` acts as a protective layer around the exclusive `__TiagoAPIInstance` instance. The primary role

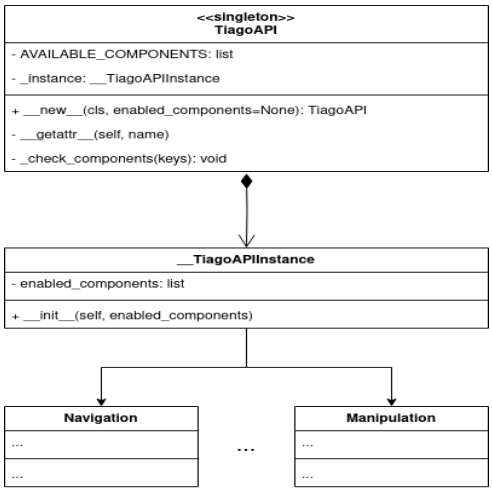

Figure 3: TiagoAPI UML Class Diagram

of the outer `TiagoAPI` class is to ensure the existence of only one `__TiagoAPIInstance` instance, achieved through the `_instance` class attribute and the `new` method. Additionally, the interaction between `__TiagoAPIInstance` and its components is designated by a relationship, given its role in orchestrating these components.

A critical focus of this thesis was the conception and integration of TiagoAPI, bridging the gap between a robot's inherent hardware, particularly its sensors and actuators, and the layered functionalities built upon them. We devised a structured and hierarchical architecture based on efficient grouping for optimal efficiency, modularity, and adaptability. Figure 4 illustrates this architectural design.

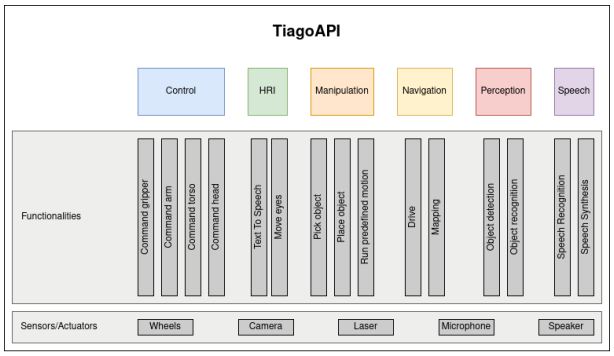

Figure 4: TiagoAPI Layered Architecture

Within this architecture, we integrated the following key components:

- Control: This component oversees and manages the primary hardware operations, coordinating the robot's lower-level hardware components, including joints, wheels, and other mechanical parts. Such centralized control en-

sures that every action aligns with the robot's objectives, whether it involves subtle joint movements or navigation through space using wheels.

- Human–Robot Interaction (HRI): The HRI component facilitates natural and symbiotic interaction between the robot and humans. Gaze and Text To Speech (TTS) functionalities enable effective communication with users. Emphasizing HRI underscores the importance of a user-friendly robot capable of comprehending and responding to human commands and inquiries.
- Manipulation: Focused on the robot's physical interaction with its environment, this component handles tasks like object picking, placing, and grasp pose estimation. These capabilities enable the robot to interact with and manipulate objects accurately and effectively.
- Navigation: An essential aspect of a robot's autonomy, this component covers functionalities related to mapping, localization, and autonomous movement. A sophisticated navigation system enables the robot to operate in diverse environments, avoiding obstacles and reaching specific destinations.
- Perception: This component forms the foundation for the robot to interpret and respond to its surroundings actively. It includes advanced functionalities such as environment modeling, tracking, object detection, and individual recognition. These capabilities enhance the robot's situational awareness and enable informed decision-making based on its environment. Integrating perception and navigation is crucial for tasks like person following and object recognition.
- Speech: This component encompasses speech-related functionalities like speech recognition and language understanding. Effective speech interaction enhances the robot's communication capabilities. A robust dialogue system ensures that the robot can understand user commands and engage in meaningful conversations.

The development and refinement of TiagoAPI demanded meticulous attention to ensure its quality and reliability. A robust debugging system was established, fortified by static code analysis, unit tests, and logging tools.

Unit tests are essential for debugging and examining the API thoroughly, ensuring the precision of individual code segments, like functions. These tests, performed with specific inputs, validate the alignment of results with expectations, especially in the context of robotic software development, to prevent inadvertent errors.

Supplementing the feedback from unit tests, an advanced logging system was integrated into the API, recording system activities, events, and modifications. Developers can leverage these logs to trace the causes of unexpected issues, benefiting from timestamps, contextual details, and error notes for efficient issue resolution.

To enhance clarity, we prioritized comprehensive documentation for TiagoAPI, aiding developers in comprehending its complexities and functionalities. This documentation serves as a reliable reference guide, ensuring consistent implementation and optimal utilization.

In summary, the debugging phase was indispensable in the creation of TiagoAPI. Leveraging unit tests, logging, simulations, CI tools, and code checks, the API adheres to top-tier software practices, promising users a reliable and streamlined platform for robotic interaction.

4.2. Auto Diagnostics Service

This section discusses the development of a diagnostic system designed to work alongside the API as part of the robotic framework. Its primary purpose is to assess the robot's state during initialization or at specific intervals, enabling the dynamic adjustment of the API's active components to ensure its compatibility across various robotic platforms.

The diagnostic system relies on analyzing two crucial files, the Unified Robot Description Format (URDF) and Semantic Robot Description Format (SRDF), which provide comprehensive representations of the robot's physical and semantic attributes. By meticulously examining these files for essential elements like nodes, joints, and topics, the diagnostic system can promptly identify any deficiencies.

For instance, if a critical arm joint is missing in the URDF or SRDF files, the diagnostic system swiftly disables the corresponding component within the API, such as the manipulation component. This action alerts programmers to the unavailability of the manipulation component, encouraging them to address the issue by rectifying the missing joint or modifying their code accordingly.

The high-level workflow outlined in Listing **??** showcases the key steps involved in the diagnostic system's operation. It involves initializing the robot and the diagnostic system, loading relevant files, performing checks, and enabling dynamic updates to the API components when necessary.

In conclusion, the integration of this diagnostic system enhances the adaptability of the API, enabling smooth transitions across different robotic platforms. By proactively evaluating the robot's

health and responding intelligently to its configuration, this system significantly contributes to the API's resilience and reliability across diverse robotic platforms.

4.3. API CMD Tool

The API CMD Tool acts as a mediator between developers and the TiagoAPI, providing an intuitive platform for interacting with the robot's features. Simplifying the complexities of the ROS ecosystem, this tool, encompassing the `interactive_node.sh` shell script, enables real-time experimentation through an IPython interface, enhancing user experience with improved debugging and auto-completion functionalities.

The script verifies the accessibility of the `actions_tiago` ROS package and the presence of IPython on the system, granting direct access to the TiagoAPI. Additionally, the `tiago_api_class_node` Python module establishes a tailored ROS node, facilitating seamless interactions with the robot during the IPython session.

Overall, the API CMD Tool underscores the project's dedication to accessibility and user-friendliness, catering to both seasoned roboticists and newcomers. By providing an efficient means of interaction, it ensures swift development, testing, and iterative processes, promoting the accessibility of the robot's advanced capabilities.

5. Experiments

The TiagoAPI's performance underwent a comprehensive evaluation, involving diverse experiments in both simulated and real-world settings. Significantly, invaluable insights were derived from the utilization of the TiagoAPI in the RoboCup competition. The RoboCup challenges served as an exceptional testing ground, allowing for a comprehensive evaluation of the TiagoAPI's capabilities in a dynamic and competitive environment that closely simulated real-world complexities.

5.1. Simulated Environments

The TiagoAPI was rigorously tested in simulated environments, primarily using Gazebo, an open-source 3D robotics simulation tool. The simulation environment, modeled after the IS-RoboNet@Home's apartment infrastructure, provided an effective platform for evaluating the robot's performance. (Figure 5)

During the testing phase, the TiagoAPI's capabilities were assessed through various ROS tutorials, notably the Pick & Place Demo and the Planning with Octomap Demo.

The Pick & Place Demo showcased the robot's ability to perform tabletop pick and place tasks, integrating ArUco for object reconstruction and MoveIt! for seamless object manipulation.

Figure 5: ISRoboNet@Home Testbed Simulated

The Planning with Octomap Demo highlighted the TiagoAPI's proficiency in navigating complex environments and planning collision-free trajectories by effectively incorporating Octomap into MoveIt!.

These evaluations significantly deepened the understanding of Tiago's operational characteristics and limitations.

5.2. Real-World Scenarios

During the TiagoAPI evaluation, real-world scenarios supplemented simulated environments, enhancing understanding of its practical applicability. An exemplar was the 'Receptionist' task from RoboCup, embodying complex human-robot interaction requirements. The task's demands, illustrated in Figure 6, highlighted the need for seamless integration, precise perception, and efficient navigation.

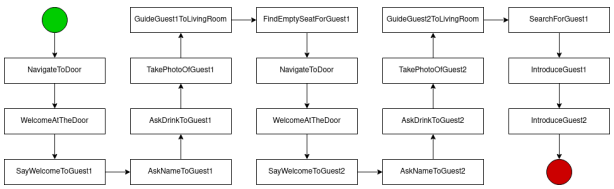

Figure 6: RoboCup Task Receptionist - State Machine for Successful Execution

A state machine, representing operational phases and transitions in response to stimuli, facilitated the task. For instance, 'FindEmptySeatForGuest1' leveraged TiagoAPI's perception for seat analysis. Notably, the majority of functions in the state machine employed the TiagoAPI, showcasing its seamless integration for intended behaviors.

Witness the 'Receptionist' task in action via this link: `https://www.youtube.com/watch?v=Q7XkN0Y095o`.

6. Conclusions

In this research, the TiagoAPI was created for the TIAGo mobile service robot, aiming to simplify its control and enhance its functionalities. This comprehensive and user-friendly solution facilitates interaction for both experienced developers and those with limited robotics knowledge, thus lowering the entry barrier. The incorporation of a versatile diagnostic system ensures the adaptability of the API across various robotic platforms, reinforcing its future-ready nature. The integration of the API CMD Tool further promotes ease of use, fostering an environment conducive to swift development and testing. This research emphasizes the significance of accessibility and adaptability in the realm of robotics, paving the way for a more inclusive and intuitive robotic ecosystem that can seamlessly integrate into everyday life. The ongoing evolution of the TiagoAPI is anticipated to align with technological advancements and the changing needs of the robotics community, encouraging collaborative research and development for the advancement of the field.

6.1. Future Work

The TiagoAPI presents a strong foundation for robotic functionalities, yet there is room for further refinement and exploration. One significant challenge is ensuring the API's ability to grasp diverse tasks. Utilizing transferable learning systems, the API could understand and adapt to various tasks based on its prior knowledge.

Future enhancements for TiagoAPI could focus on:

1. Enhanced Transferability Across Robotic Platforms: Expand adaptability to integrate seamlessly with a broader range of robotic platforms, reducing the need for custom adaptations and configurations during transitions between different robot types and models.

2. Improved Diagnostics and Feedback: Enhance real-time feedback and error correction for a smoother user experience. Providing comprehensive insights into operational status can facilitate efficient troubleshooting and refined interaction with the system.

3. Intuitive User Interface and Experience: Incorporate a more user-friendly interface, including visual programming tools, drag-and-drop functionalities, and context-aware help systems, catering to both expert and novice users, thereby streamlining the robot programming process.

In conclusion, while the current TiagoAPI has simplified robot programming, future versions could offer even more sophisticated, user-friendly functionalities. By addressing the identified gaps and building on the existing framework, the TiagoAPI can become a more powerful tool for roboticists and developers.

Acknowledgements

I'm grateful for the opportunity to contribute to the project, guided by Prof. Pedro U. Lima and the SocRob@Home team. Their support has deepened my passion for robotics. Postdoctoral Researcher Carlos Azevedo's unwavering guidance was crucial in the challenging phases, leading to the project's success. My heartfelt thanks to my family and friends for their unwavering encouragement. Thank you.

References

[1] A. Angerer, A. Hoffmann, A. Schierl, M. Vistein, and W. Reif. The robotics api: An object-oriented framework for modeling industrial robotics applications. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4036–4041, 2010.

[2] M. A. K. Bahrin, M. F. Othman, N. H. N. Azli, and M. F. Talib. Industry 4.0: A review on industrial automation and robotic. *Jurnal teknologi*, 78(6-13), 2016.

[3] T. Belpaeme, J. Kennedy, A. Ramachandran, B. Scassellati, and F. Tanaka. Social robots for education: A review. *Science robotics*, 3(21):eaat5954, 2018.

[4] N. Bostrom. Superintelligence: Paths, dangers, strategies. *Oxford University Press*, 2014.

[5] R. Bourqui, P. Fankhauser, A. Guignard, and M. Hutter. A survey of robot operating system (ros). In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3376–3383. IEEE, 2014.

[6] A. Burke and J. Lasenby. Using the microsoft robotics developer studio as a simulation tool for multi-robot systems. *International Journal of Computer Applications*, 29(12):22–28, 2011.

[7] E. Davies, G. Neumann, and J. Ziegler. A review of domestic service robots: Market, technology and future prospects. *Robotics and Autonomous Systems*, 76:16–29, 2016.

[8] M. Fink, H. Gassen, and U. Reiser. Domestic service robots: A market overview. *International Journal of Advanced Robotic Systems*, 7(4):365–378, 2010.

[9] P. Fitzpatrick, G. Metta, and L. Natale. Towards long-lived robot genes. *Robotics and Autonomous Systems*, 56(1):29–45, 2008.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[11] X. Gao, S. Zhang, H. Li, and J. Wang. Design and implementation of an api-based control framework for intelligent robots. *IEEE Access*, 5:22462–22473, 2017.

[12] F. Gomez and N. Patel. Integrating orocos and ros: A study on seamless robot software platforms. *Robotics and Computer-Integrated Manufacturing*, 45:52–61, 2023.

[13] N. He, Z. Chen, Q. Zhang, and J. Li. An api-based approach for integrating multiple sensors in robotic applications. *IEEE Access*, 6:77207–77216, 2018.

[14] R. Hernandez and P. O'Donnell. *Robotic Software Frameworks: Past, Present, and Future*. Springer, 2020.

[15] N. G. Hockstein, C. Gourin, R. Faust, and D. J. Terris. A history of robots: from science fiction to surgical robotics. *Journal of robotic surgery*, 1(2):113–118, 2007.

[16] G. Hu, W. P. Tay, and Y. Wen. Cloud robotics: architecture, challenges, and applications. *IEEE Network*, 26(3):21–28, 2010.

[17] M. Jackson. Microsoft robotics studio: A technical introduction. *IEEE Robotics Automation Magazine*, 14(4):82–87, 2007.

[18] E. Jones, T. Oliphant, and P. Peterson. *SciPy and NumPy: open source scientific tools for Python*. Elsevier, 2001.

[19] Q. Khan, H. Ahmed, M. Usman, N. Khan, W. Khan, and N. Ahmad. A comparison of c++ and python in ros: a case study. *International Journal of Advanced Robotic Systems*, 16(6):1729881419864261, 2019.

[20] S. Khan, M. Z. Qureshi, N. Ahmad, and L. Zaman. A survey of robotic apis: Design and implementation. *IEEE Access*, 7:124596–124613, 2019.

[21] R. D. King, J. Rowland, S. G. Oliver, M. Young, W. Aubrey, E. Byrne, M. Liakata, M. Markham, P. Pir, L. N. Soldatova, et al. The automation of science. *Science*, 324(5923):85–89, 2009.

[22] S. Kounev, T. Hildenbrand, and F. Leymann. A survey of middleware for the internet of things. *IEEE Internet Computing*, 15(6):50–57, 2011.

[23] J. Lee, T.-W. Kim, and S.-H. Kim. Domestic robots: Status and prospects. *International Journal of Advanced Robotic Systems*, 9(4):1–15, 2012.

[24] P. U. Lima, C. Azevedo, E. Brzozowska, J. Cartucho, T. J. Dias, J. Gonçalves, M. Kinarullathil, G. Lawless, O. Lima, R. Luz, et al. Socrob@ home. *KI-Künstliche Intelligenz*, 33(4):343–356, 2019.

[25] P. Lin, G. Bekey, and K. Abney. Autonomous military robotics: Risk, ethics, and design. Technical report, California Polytechnic State Univ San Luis Obispo, 2008.

[26] Y.-J. Lin, S.-M. Huang, C.-H. Tsai, P.-H. Chen, and C.-M. Chen. Design and implementation of a robotic api for enhanced usability and extensibility. *IEEE Access*, 8:162315–162325, 2020.

[27] LinkedIn. Pros and cons of using ros (robot operating system) in robotics, 2023. Accessed on September 30, 2023.

[28] J. Liu, Y. Liang, and Q. Huang. Natural language processing for robotic systems: A review. *IEEE Access*, 4:4181–4192, 2016.

[29] G. Metta, P. Fitzpatrick, and L. Natale. Yarp: Yet another robot platform. *International Journal of Advanced Robotic Systems*, 3(1):043, 2006.

[30] R. Murphy. An introduction to artificial intelligence. *Elsevier*, 2015.

[31] L. Natale, F. Nori, G. Sandini, G. Metta, and D. Vernon. The icub platform: A tool for studying intrinsically motivated learning. *Frontiers in Robotics and AI*, 3:26, 2016.

[32] F. Perez and B. E. Granger. Ipython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, 2007.

[33] M. Quigley, K. Conley, B. Gerkey, et al. Ros: An open-source robot operating system. *ICRA Workshop on Open Source Software*, 3(3.2):5, 2009.

[34] A. Rosenfeld and B. Green. Orocos: An in-depth analysis of a modular robot control framework. *Journal of Robotic Systems*, 39(4):321–334, 2022.

[35] Y. Shang, S. Zhang, X. Gao, and H. Li. Api-based integration of robotic systems: A survey. *IEEE Access*, 4:5348–5359, 2016.

[36] J. Shi, X. Gao, H. Li, and S. Zhang. Integration of robotic systems using apis: A survey. *IEEE Access*, 7:124539–124550, 2019.

[37] G. Van Rossum. An overview of the python language. *Computer*, 23(11):31–41, 1990.

[38] L. Wang and J. Smith. Modularity in robot software design: Benefits and challenges. In *Proceedings of the 8th International Conference on Robotics and Automation*, pages 112–119, 2021.