TÉCNICO LISBOA

API for an Autonomous Mobile Service Robot

Raphaël de Araújo Colcombet

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisor: Prof. Pedro Manuel Urbano de Almeida Lima

Examination Committee

Chairperson: Prof. Paolo Romano
Supervisor: Prof. Pedro Manuel Urbano de Almeida Lima
Member of the Committee: Prof. António Manuel Ferreira Rito da Silva

November 2023

This work was created using LaTeX typesetting language
in the Overleaf environment (www.overleaf.com).

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

I am truly honored to have had the chance to make a meaningful contribution to this project.

Prof. Pedro U. Lima and the entire SocRob@Home team have been truly remarkable. Despite being a working-student abroad, their feedback, fellowship and assistance have significantly enriched my experience. This project has further solidified my interest in robotics and its challenges.

I am also deeply grateful to Postdoctoral Researcher Carlos Azevedo, whose consistent support and guidance were indispensable, particularly during the demanding phases of the project. His expert advice significantly contributed to its successful completion.

Lastly, I extend my sincere thanks to my beloved family and friends for their continuous encouragement and constant support throughout this fulfilling journey.

Thank you.

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 effort underscores the indispensability of user-centric robotic software frameworks, reinforcing the central idea that they constitute the key to the work. 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;

Resumo

A presente tese explora as estruturas fundamentais e os resultados da API SocRob@Home para um Robot de Serviço Móvel Autónomo, realçando a utilidade de linguagens de programação de propósito geral na criação de uma camada de abstração útil. O objetivo principal desta abstração é otimizar a configuração, operação e controlo do robô, abstraindo os detalhes tecnológicos complexos, proporcionando uma interface mais amigável para o desenvolvimento de futuras aplicações robóticas.

A TiagoAPI, central para este projeto, oferece uma abordagem orientada a objetos projetada para uma integração perfeita com uma vasta gama de plataformas robóticas. Simplifica processos robóticos complexos em comandos compreensíveis, permitindo uma operação eficiente e versátil do robô. A API está enraizada em paradigmas modernos de software e oferece um conjunto abrangente de funcionalidades, como navegação avançada, manipulação de objetos, capacidades perceptivas, processamento linguístico e formulação estratégica de tarefas.

Este trabalho sublinha a necessidade de estruturas de software robótico centradas no utilizador. Ao tornar componentes robóticas complexas, como navegação e gestão de tarefas, mais acessíveis através da TiagoAPI, a pesquisa contribui para a redução da lacuna entre as capacidades robóticas e a acessibilidade do utilizador, visando uma base de utilizadores diversificada com proficiência técnica variada.

Palavras Chave

Arquitetura robótica; API de robô; API; ROS;

Contents

List of Figures

Listings

Acronyms

AI	Artificial Intelligence
API	Application Program Interface
CCR	Concurrency and Coordination Runtime
DDS	Data Distribution Service
HRI	Human–Robot Interaction
IPC	Inter-Process Communication
ISR	Institute for Systems and Robotics
IST	Instituto Superior Técnico
KDL	Kinematics and Dynamics Library
RDS	Microsoft's Robotics Developer Studio
ROS	Robot Operating System
RTT	Real-time Toolkit
SRDF	Semantic Robot Description Format
TTS	Text To Speech
UML	Unified Modeling Language
URDF	Unified Robot Description Format
VPL	Visual Programming Language

1

Introduction

Contents

Robotics has become a significant tool in various fields, including industry [1], science [2], the military [3], health [4], and education [5]. As an interdisciplinary field, the development of robotics often requires expertise in combining multiple fields such as engineering, computer science, and materials science [6].

In the current age of information revolution, which is characterized by the rapid development and widespread adoption of digital technologies, the development of robotics has been closely tied to the advances made. As a result, robotics has become an important part of many industries and has the potential to revolutionize the way people live, work, and interact with technology [7].

Domestic robots are a rapidly growing field within the broader field of robotics [8]. These robots are designed to perform a variety of tasks within the home, including cleaning, cooking, and providing companionship [9]. As the capabilities of domestic robots continue to improve and their costs decrease [10], they are becoming increasingly popular among households around the world.

SocRob@Home [11] is the contribution to that progress from Institute for Systems and Robotics (ISR) research group, at Instituto Superior Técnico (IST) since 1998, in the world's leading scientific event on Artificial Intelligence (AI) and Robotics, RoboCup. As part of this project, researchers are encouraged to work together as an engineering team to solve a range of different problems from hardware to software (e.g., wireless communications, navigation, control, electronics). SocRob@Home has a prestigious track record in scientific competitions, making significant strides in RoboCup 2018, ERL Smart Cities 2019, and RoboCup 2021. The team's recent triumph at RoboCup 2023 [12] in Bordeaux, securing the second place, Figure 1.1, marks a notable advancement and the best performance to date, showcasing dedication to elevating the field of robotics. This historic victory is a testament not only to the current members of the team but also to the contributions of previous students who have been part of the project.

Within the scope of this project, the robot used is the TIAGo mobile service robot, a creation of PAL Robotics. TIAGo is a mobile manipulator robot, it has a body and limbs similar to a human's, making it suited for tasks that require interactions with the environment, such as picking up and moving objects, or interacting with people.

Figure 1.1: RoboCup 2023 Award Ceremony - SocRob@Home

1.1 Context

An Application Program Interface (API), is a set of guidelines and standards for accessing software applications or tools [13]. A robot API system is a type of API specially designed to allow different software programs to communicate with each other and share data. This is particularly useful in the technology field, because multiple systems may be required to work together to perform complex tasks. For example, a robot API could enable a robot to access information from a sensor in order to navigate a physical space, or to communicate with a user through a natural language processing system [14]. By using a robot API, developers can easily integrate a wide range of functionality into their systems, allowing for more advanced and flexible behavior. Many of the robot's functionalities are implemented using Robot Operating System (ROS) and to use them normally, one must know which packages to launch, which nodes to subscribe to, and what types of messages to use. This process is not very user-friendly and consumes a significant amount of time for those who need to use these functionalities. For this reason, the API will also reduce costs on building and implementing individual components. Additionally, a robot API can provide a consistent and standardized interface, making easier for developers to understand and use various components that make up a robotic system [15]. A robot API can be thought of as an abstraction layer, in that it provides a simplified interface for accessing and using the underlying

functionality of a robot system [16]. This allows developers to focus on the high-level goals of their project, rather than having to worry about the details of how individual components work.

The implementation of an API for an autonomous mobile service robot presents a number of challenges and opportunities. On the one hand, the integration of various sensors and software systems can be complex and require a deep understanding of the underlying technologies. On the other hand, an API can provide a platform for developers to create innovative applications that take advantage of the capabilities of the robot. By providing a clear and easy-to-use interface, an API can facilitate the development of new applications and features that can greatly enhance the usefulness and capabilities of the robot.

For non-developers, the use of a robot API to integrate an algorithm or interact with a ROS component can be easily done without having a deep understanding of software development. This API will not only aid in the organization of the software but also enforces modularization and compartmentalization. These design principles assist immensely in the seamless transfer of functionalities between various robot systems. However, a central challenge remains: developing a universally applicable and reliable robot API. Given the nascent and swiftly evolving nature of robotics, the criteria and functionalities of these API are subject to change. This necessitates that developers stay abreast with changes and adapt their code proactively.

1.2 Motivation

The main aim of this thesis is to design and implement a programming interface that provides access to all the functions and algorithms of the TIAGo mobile service robot, such as navigation, manipulation, grasping, speech understanding, and others. The idea behind this API is to encapsulate and document all the functionalities in a single library in such a way that other programmers can easily invoke each of them, with the required arguments, in an uniformized and coherent manner, without requiring deep knowledge of robotics. This will enable them to combine the functions together in a task. A crucial step in creating this interface is to identify and classify the implemented functions and algorithms.

In addition to this, the research aims to make the API open-source, allowing the community to benefit from and expand upon its capabilities.

A further objective of this research is to gain a deeper understanding of the different robot subsystems and improve expertise in integrating robot systems through the development of the API.

1.2.1 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 TiagoAPI 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

Contents

This chapter takes an in-depth look at the foundational tools and platforms that form the SocRob@Home 2023 project. The selection of Python as the primary programming language, the multifaceted capabilities of the ROS, and the features and significance of the TIAGo robot are examined. These tools, chosen for their versatility and relevance, collectively serve as the backbone of our development endeavors in the realm of domestic robotics.

2.1 Python

Python, developed by Guido van Rossum in 1991, is often chosen as the programming language for a solution because it is a flexible, high-level language that is easy to learn and use [17]. Python has a large and active community of users and developers, and it offers a wide range of libraries and frameworks that can be used for a variety of purposes, including robotics. Additionally, Python is an interpreted language, which means that it can be run on a wide range of platforms without the need for compilation [18]. This makes it easy to write and test code quickly, and to deploy it on different devices and systems [19]. These attributes collectively contribute to Python's preeminence as perhaps the most widely used language in research.

ROS can be used with a variety of programming languages, including Python and C++ [20]. Both Python and C++ have their own strengths and weaknesses, and the choice of which language to use depends on the specific requirements and preferences of the project. Generally, Python excels in prototyping and swift development, whereas C++ shines in performance-intensive applications due to its potency. ROS itself is written in C++, and many of the core libraries and tools are also written in C++, so using C++ can provide tighter integration with ROS [21]. However, Python is also well supported in ROS, and it offers many advantages, such as a large and active community, a wealth of libraries and frameworks such as rospy, MoveIt, and OpenCV, and ease of use.

Ultimately, the decision to use Python for this project stemmed from several considerations tailored to its specific objectives. One primary need of the project was seamless integration with our existing systems, where our team predominantly employs Python for development tasks. The project's goal was to foster an environment conducive to quick iterations and modifications, and Python's extensibility, with its rich ecosystem, aptly supports this. Additionally, the API designed for this project does not

necessitate complex algorithmic computations, but merely serves as a liaison, making the computational intensity of the language less of a concern. Python's inherent readability and simplicity further play a crucial role in curtailing the learning curve for new developers joining the team.

2.2 ROS

ROS is a popular framework for building and managing robotic systems. Developed by Willow Garage in 2007, ROS provides a set of tools and libraries for constructing distributed systems and implementing common robot functionality. It also offers a hardware abstraction layer, and provides a set of tools and libraries that support the quick and easy construction of distributed systems, as well as a broad collection of capabilities for implementing common robot functionalities [22]. Additionally, ROS is supported by a large and active community, with focus on integration and documentation.

At its core, ROS comprises four fundamental components, each playing a vital role in enabling the development and management of robotic systems:

- Messaging: ROS provides a robust publish-subscribe communication system that facilitates seamless interaction between various components of a robot. This communication mechanism supports the construction of distributed systems and allows for a modular design approach [22]. For instance, different parts of a robot, such as sensors, actuators, and control algorithms, can communicate efficiently through ROS's messaging system, which includes Inter-Process Communication (IPC) capabilities. This IPC feature ensures that data exchange between different processes within a robot occurs efficiently and reliably. By leveraging IPC, ROS enables modular and parallelized development, enhancing flexibility and reusability in robot design.

- Utilities: ROS offers a rich set of utilities that simplify the configuration, deployment, and maintenance of robotic systems. These utilities encompass a wide range of functionalities, including configuration management, system introspection, debugging tools, visualization tools, logging mechanisms, testing frameworks, and the ability to start or stop distributed systems [20]. These tools make it easier for developers to manage and troubleshoot complex robotic setups, ultimately accelerating the development process.

- Libraries: ROS includes a comprehensive collection of libraries that implement essential robot functionalities. These libraries cover diverse domains, such as mobility (e.g., motion planning and control), manipulation (e.g., robotic arm control), and perception [22] (e.g., computer vision and sensor data processing). By leveraging these libraries, developers can expedite the development of specific robot capabilities, saving time and effort while benefiting from established best practices in various robotics domains.

- Ecosystem: The ROS ecosystem is defined by a vibrant community of developers and users actively contributing to and utilizing the framework. This community support is instrumental in advancing ROS and maintaining its relevance. Through the ROS community, developers gain access to a wealth of knowledge, documentation, and collaborative resources. Moreover, the ROS ecosystem includes a vast repository of ROS packages, accessible through the official `ros.org` website [20], offering ready-made solutions and pre-built modules for a wide array of robotic applications. This extensive collection of packages simplifies the integration of third-party hardware and software components into ROS-based systems.

ROS has been used in a variety of different applications, including mobile robotics, aerospace, manufacturing, and service robotics [22]. In mobile robotics, ROS has been used to build and control a wide range of platforms, including ground vehicles, aerial drones, and underwater robots [20]. In aerospace, ROS has been used to build and manage autonomous spacecraft, including the NASA Mars rover Curiosity. In manufacturing, ROS has been used to build and control robotic manipulators for tasks such as welding, painting, and assembly. In service robotics, ROS has been used to build and control robots for tasks such as home assistance, healthcare, and disaster response.

ROS also provides a unified set of APIs for common tasks such as motion planning, sensor processing, and communication, allowing developers to focus on building the core functionality of their robots [20]. This can save significant development time and effort, as developers do not need to reinvent the wheel for basic functionality [22]. Additionally, the large and active ROS community is a valuable resource for developers, providing access to extensive documentation, support, and a wealth of existing packages and libraries [20].

However, it's essential to acknowledge potential drawbacks when utilizing ROS. One notable concern pertains to ROS's reliance on a publish-subscribe messaging

model. This model can result in elevated overhead and latency, particularly in situations involving substantial data transfers or frequent requests [22]. Furthermore, adopting ROS may entail added complexity due to the necessity of incorporating supplementary libraries and tools. Moreover, newcomers might encounter a learning curve when getting acquainted with ROS [20].

Overall, ROS is a powerful and widely-used framework for building and managing robotic systems. Its broad compatibility with diverse hardware platforms, comprehensive set of libraries and tools, and vibrant developer community render it an appealing option for numerous developers. Nevertheless, it is imperative to conduct a thorough assessment of the potential drawbacks associated with ROS and assess its suitability for a specific application [23].

2.3 TIAGo Robot

The utilization of robotic platforms plays a pivotal role in the realization of the SocRob@Home 2023 project's research objectives. Within this scope, we employ the TIAGo mobile service robot (see Figure 2.1), a creation of PAL Robotics, renowned for its capabilities and versatility in addressing the multifaceted challenges of domestic robotics.

Figure 2.1: PAL Robotics TIAGo Steel Edition Robot

TIAGo stands as a mobile manipulator robot, featuring a humanoid-like body and limbs that closely mimic human characteristics. This humanoid design enhances its suitability for tasks that necessitate intricate interactions with the environment, including the manipulation of objects and engaging with individuals. The robot boasts a

comprehensive suite of sensors, a robust processing unit, and sophisticated software components, making it an ideal choice for the multifarious research domains encapsulated within our project.

Key attributes of the TIAGo Steel robot, the variant employed in our research, include a differential-drive base, a front laser range-finder, and rear sonars. These sensors are instrumental in tasks such as mapping, navigation, and obstacle avoidance, enabling the robot to navigate and operate within domestic environments effectively. The robot has a 7-degree-of-freedom arm, equipped with a parallel gripper as an end-effector, and a lifting torso. These features empower TIAGo to perform tasks that require reaching and grasping objects, including pick and place operations. Furthermore, the inclusion of a speaker and a stereo microphone enhances its capabilities in human-robot interaction, facilitating natural and effective communication with users. In the realm of perception, TIAGo is equipped with an RGB-D camera positioned in its head, enabling functions such as object detection and localization, people tracking, obstacle perception, and visual servoing.

The adaptability and configurability of TIAGo are pivotal for our research endeavors. This mobile manipulator robot is designed to continuously adapt to evolving research needs, making it an invaluable asset for our exploration of various domains within domestic robotics. Its seamless integration of perception, navigation, manipulation, and human-robot interaction skills out of the box aligns perfectly with the project's objectives. Whether it be for research projects or participation in robotics competitions, TIAGo is capable of excelling in fields such as ambient-assisted living, healthcare, and light industry.

In summary, the TIAGo mobile service robot is the central tool for the research efforts in the SocRob@Home 2023 project. The humanoid-inspired structure, inclusive sensor range, and strong functionalities enable effective resolution of the intricate hurdles within domestic robotics. Exploring the nuances of mapping, localization, navigation, perception, manipulation, decision-making, and human-robot interaction, TIAGo has proven to be indispensable.

3

Related Work

Contents

In the evolving realm of robotic technologies, understanding the previous work is pivotal for advancing innovations and bridging gaps between complexities. This chapter delves into pertinent research related to this thesis dissertation, shedding light on significant endeavors in the domain of robot APIs, especially those that transcend the limitations of specific hardware configurations. By providing an in-depth examination of studies that offer valuable insights into robot APIs in varying scenarios, our aim is to elucidate the techniques adopted, the challenges faced, and the notable outcomes achieved. This exploration not only facilitates a comprehensive understanding of the prior efforts but also accentuates the distinctiveness and potential benefits of our unique approach, which emphasizes simplicity and versatility by decoupling software intricacies from hardware constrains. Through this lens, we invite the reader to appreciate the broader context within which our proposed solution operates, and the strides made in making robotic control more accessible and adaptable.

3.1 The Robotics API

In [24], Angerer presents the Robotics API, designed to modernize industrial robot programming, traditionally tied to proprietary languages rooted in older imperative languages like ALGOL or Pascal. Such traditional languages, while powerful in their niche, lack flexibility found in general-purpose languages.

The Robotics API fills this gap with its object-oriented approach, allowing developers to model real-world entities as software objects. This not only improves clarity but also promotes code reuse, reducing development overhead. Core to the API is its ability to model the physical world, including spatial relationships and diverse workpiece representations, with around 70 classes catering to various devices.

A standout feature is its unique command handling, inspired by the ”Command pattern” in [25]. This allows for the easy combination of multiple commands, considering intricate timing sequences.

Built on Microsoft’s C# and .NET framework, the API offers protection against common programming issues. Its design ensures rapid and reliable software development, evidenced by the error-free welding example.

The practicality of the API is confirmed by its deployment in controlling two KUKA lightweight robots. Future updates look to broaden its scope, including better sensor integration and improved error handling. The Robotics API stands as a notable innova-

tion, aiming to make advanced robotic features more accessible, especially to smaller enterprises.

In essence, the Robotics API underscored the significance of marrying traditional robotics with modern programming paradigms, inspiring TiagoAPI to mirror such principles.

3.2 YARP: Yet Another Robot Platform

In the diverse landscape of robotic software frameworks, the YARP platform, short for ”Yet Another Robot Platform,” holds a special place. Emerging from a need to connect heterogeneous robot parts and computational devices, YARP alleviates challenges of real-time communication and data sharing among robots, sensors, and processors [26].

In stark contrast to proprietary systems that enforce vendor-specific solutions, YARP is open-source and operates in a modular fashion. This platform promotes a ”write once, run anywhere” paradigm, a goal realized by abstracting away the intricacies of inter-device communication. It achieves this through the creation of ports - interfaces that facilitate bidirectional data exchange between processes, even if they are running on different machines [26].

The brilliance of YARP lies in its simplicity and adaptability. Rather than being a monolithic structure, it offers essential primitives required for network communications, thereby allowing developers to mold and augment the platform as per their requirements [27]. Its design is inherently object-oriented, making it seamless to encapsulate real-world robotic entities, sensors, and actuators as software components.

YARP supports middleware – essentially, intermediary software modules that facilitate communication or data sharing. This means that developers can integrate various communication protocols, be it ROS, Data Distribution Service (DDS), or even custom solutions, without altering the core logic of their applications [26].

In practical applications, YARP has demonstrated its mettle, aiding in the control and coordination of a plethora of robotic platforms, from humanoid robots to manipulative arms [28]. The platform’s flexibility and extensibility underscore its potential to continue being a staple in the robotic software ecosystem, bridging gaps between hardware, real-time constraints, and developers’ visions.

The recognition of YARP’s capabilities, combined with its practical application in

real-world situations, played a role in the creation of the TiagoAPI. Using the same principles of YARP as a reference, the goal was to develop an API that addresses specific needs and aligns with developers' requirements, ensuring that the TiagoAPI remains reliable and adaptable.

3.3 Microsoft's RDS: Robotics Developer Studio

Microsoft's Robotics Developer Studio (RDS) was born out of a vision to offer an integrated environment that simplifies the creation, testing, and deployment of complex robotic applications [29]. Its introduction heralded a shift towards an object-oriented approach, capitalizing on the modularity, reusability, and organization offered by such a paradigm [30].

A key feature of RDS is its Visual Programming Language (VPL) which provides a graphical interface for designing and observing the flow of robotic applications. The ease of drag-and-drop, coupled with an intuitive visual representation, abstracts away the intricacies of code and promotes rapid prototyping [29].

Beyond VPL, RDS showcases a robust set of services that champion a decentralized approach. Each service, encapsulating specific functionalities like sensor interfacing, motion planning, or diagnostics, operates independently, communicating with other services as needed [31]. This modular approach aligns with object-oriented principles, allowing developers to represent robotic systems and components as software services.

Its Concurrency and Coordination Runtime (CCR) stands out, addressing the challenges of concurrent operations in robotics. By facilitating responsive, asynchronous message-passing, the CCR ensures that robotic systems can adeptly handle simultaneous tasks, making the most of multi-core processors [29].

The transferability of RDS is further emphasized by its simulator, allowing developers to test and refine their applications in a risk-free virtual environment before deploying to real-world hardware. By mirroring the complexities of the physical world, this simulator accelerates development, minimizes errors, and boosts overall confidence in robotic solutions [30].

Drawing upon the vast ecosystem of Microsoft's software products and the .NET framework, RDS amalgamates a rich set of tools, libraries, and protocols, making it a beacon for democratizing robot development. Through its cohesive design and the inte-

gration of industry best practices, RDS serves as an exemplary model for constructing robotic API that are both accessible and transferable across domains [31].

In essence, the pioneering advancements and principles of RDS were instrumental in shaping TiagoAPI's vision and strategy. Through its holistic design and industry-leading practices, RDS illuminated the path for creating robotic APIs that are robust, intuitive, and versatile.

3.4 OROCOS: Open Robot Control Software

The open-source software landscape for robot control has witnessed notable solutions that emphasize modularity and real-time operation. One such prominent framework is OROCOS, the Open Robot Control Software project [32]. Originating as a solution tailored for modularity and real-time execution in robotic systems, OROCOS has epitomized the modular design philosophy and has significantly influenced robotic software development strategies across the spectrum [33].

Written in C++, OROCOS houses an array of advanced machine and robot control libraries. As it expanded, it also branched out into various middleware and tooling avenues, making it not just a monolithic tool, but an ecosystem for robotic software development [34]. OROCOS's modular and component-based architecture stands testament to the importance of creating intuitive and flexible platforms for developers, a challenge that the TiagoAPI thesis aims to address. With tools such as the Real-time Toolkit (RTT) and the Kinematics and Dynamics Library (KDL), OROCOS provides an environment wherein robot operations are not only timely but also adaptable to varying control flows.

Of particular relevance to the TiagoAPI's resource-oriented approach is OROCOS's design philosophy. By allowing each component, whether for sensor interfacing or motion calculations, to operate autonomously yet collaboratively, OROCOS mirrors the principles of viewing functionalities as distinct resources. This inherent modularity in OROCOS showcases the benefits and potential challenges of constructing a cohesive unit of software components, insights that could be foundational for TiagoAPI's development strategy [33].

Moreover, OROCOS's emphasis on integration, especially with platforms like the ROS, offers invaluable lessons on ensuring transferability and seamless compatibility [35]. Such insights can further enrich TiagoAPI's goal of integrating with existing ROS

components, elevating its utility for developers accustomed to the ROS ecosystem.

In retrospect, OROCOS's groundbreaking methodologies and design choices serve as a guiding light for endeavors that seek to simplify robotic interactions, making them accessible to a wider developer audience [34]. As TiagoAPI's vision and strategy evolve, reflecting upon the lessons from industry benchmarks such as OROCOS can provide pivotal directions and ensure the creation of a robust, user-friendly, and versatile robotic interface.

4

Proposed Solution

Contents

In this thesis, we present a solution to create an interface for defining the functions and algorithms of TIAGo. For that, we propose a resource-oriented approach to create an API that exposes robotic resources in an unified and agnostic manner. This API will simplify the process of interacting with the underlying system, making it more accessible to developers of different backgrounds and skill levels. In the following sections, we will describe the created solution.

4.1 Description

The proposed solution centers around a novel interface tailored for TIAGo robot, aiming to enhance its functionality definition and control mechanism. This interface seeks to address a challenge inherent to the robotics domain: providing an intuitive and flexible platform for developers, irrespective of their expertise or background. The underlying theme is to simplify interactions with TIAGo, ensuring that developers, both novices, and experts, can efficiently define its functionalities and control its behavior.

Our approach to realizing this vision is grounded in a resource-oriented methodology. In essence, we advocate for an API that consolidates robotic resources in a unified fashion, irrespective of their underlying intricacies. This API connects developers to TIAGo's core functionalities. By externalizing robotic resources through this API, we aim to create a seamless interaction layer, thereby rendering the underlying robotic system more accessible.

Key highlights of the proposed solution include:

- Resource-Oriented Approach: By viewing TIAGo's functionalities as resources, the solution adopts a novel perspective that makes interactions more intuitive. This methodology offers a standardized way to interact with different robotic functionalities, irrespective of their complexities.

- TiagoAPI: At the heart of the solution is the TIAGo API, designed to be both comprehensive and user-friendly. This API serves as a gateway, providing developers with access to TIAGo's core functionalities. Furthermore, its dynamic nature ensures seamless integration with existing ROS components, a major advantage for developers familiar with the ROS ecosystem.

- Accessibility: The solution is explicitly crafted to be inclusive, catering to developers of all skill levels. From the Python-based API that abstracts away the in-

tricacies of ROS to the API CMD Tool that offers an intuitive interaction platform, every aspect is tailored to simplify the user experience.

- Design: The architectural choice of the singleton design pattern ensures consistent and centralized control over TIAGo's resources. This design approach not only enhances TIAGo's performance but also ensures its predictability and reliability.
- Diagnostic System: Recognizing the importance of adaptability, the solution incorporates a diagnostic system. This system, through its meticulous analysis of the robot's configuration files, ensures that the API remains adaptable across diverse robotic platforms, highlighting the solution's future-proof nature.

In conclusion, the proposed solution introduces a paradigm shift in how developers interact with and control TIAGo. By adopting a resource-oriented approach and emphasizing accessibility and adaptability, it promises a future where robotic interactions are simplified, efficient, and inclusive.

4.2 Implementation

4.2.1 TiagoAPI

In order to enhance the accessibility for controlling the robot, we have developed a user-friendly Python API that simplifies robot control without the need for ROS. This approach aims to lower the barriers to entry for users. To illustrate the fundamental concept of this approach, consider the following example from the robot API documentation, shown in Listing 4.1.

Listing 4.1: Pseudocode for using TiagoAPI

```
1 INITIALIZE TiagoAPI as tiago_api
2 FUNCTION greeting(location):
3     CALL tiago_api.navigation.moveTo with a location as GOAL
4     CALL tiago_api.manipulation.runPredefinedMotion with "wave" as
          MOTIONNAME
```

```
5       CALL tiago_api.hri.sayAndWait with "Hello, World!" as TEXT
6   END FUNCTION
```

This interpreted approach allows the robot API to seamlessly integrate with and leverage pre-existing ROS components, eliminating the conventional need to manually configure ROS subscribers or service clients for establishing connections and communication. Moreover, the robot API operates discreetly in the background, handling these tasks behind the scenes. Additionally, it possesses the capability to initialize a ROS node using `rospy.init_node()` as necessary, creating a new node if none is currently active within the current process when it is first required.

4.2.1.A Singleton Pattern

The chosen architectural structure for developing the TiagoAPI is the singleton design pattern. This decision holds particular significance for the robot API, as it offers centralized and consistent control over the robot's resources and state. This centralized control enhances the robot's performance, predictability, and reliability. Moreover, the singleton pattern provides a global access point to the single instance of the class, simplifying interactions with the robot's resources and state for other parts of the system. This cohesive and streamlined control of the robot's behavior is crucial for achieving a synchronized and efficient robotic operation.

This architectural choice guarantees a sole instance of the `TiagoAPI` class. Nested within is the `__TiagoAPIInstance` class, embodying the actual implementation of the API's features. Figure 4.1 displays the Unified Modeling Language (UML) class diagram of the TiagoAPI.

The relationship binding `TiagoAPI` and `__TiagoAPIInstance` is compositional. In the domain of the Singleton pattern, the `TiagoAPI` serves as a protective layer around the exclusive `__TiagoAPIInstance` instance. The primary role of the outer `TiagoAPI` class is to ensure that only one `__TiagoAPIInstance` instance exists. This is done using the `_instance` class attribute and the `__new__` method. Moreover, the interaction between `__TiagoAPIInstance` and its components is designated by a relationship, given its role in orchestrating these components.

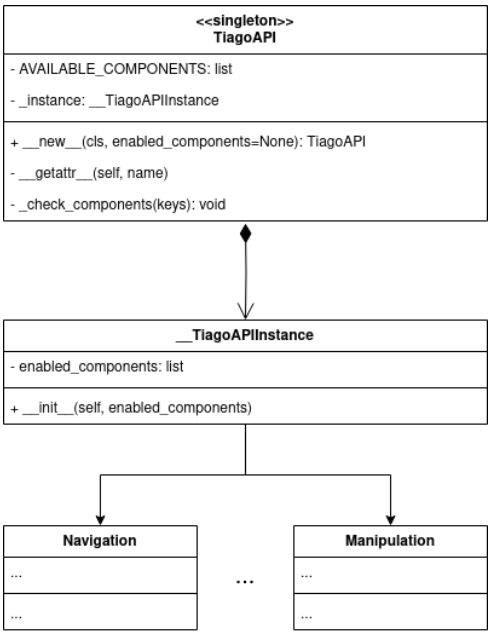

Figure 4.1: TiagoAPI UML Class Diagram

4.2.1.B Architecture and Modules

A significant aspect of this thesis was the conception and integration of the TiagoAPI, between a robot's inherent hardware, particularly its sensors and actuators, and the layered functionalities developed on top of these. A structured and hierarchical architecture was conceived. This architectural organization was based on how best to group the functionalities together for achieving optimal efficiency, modularity, and adaptability. This architectural design is depicted in Figure 4.2.

Within this architecture, the following key components were chosen and integrated:

- Control: The Control component serves to oversee and manage the primary hardware operations. It's responsible for the real-time management of the robot's lower-level hardware components, including its joints, wheels, and other mechanical parts. This centralization ensures that, whether the robot is making a subtle joint movement or navigating through space with its wheels, each action is coordinated and aligned with the robot's overarching objectives.

- Human–Robot Interaction (HRI): The HRI component plays a role in enabling the robot to interact with humans in a natural and symbiotic manner. This includes functionalities, such as gaze and Text To Speech (TTS), which are essential for effective communication with users. The emphasis on HRI underscores the im-

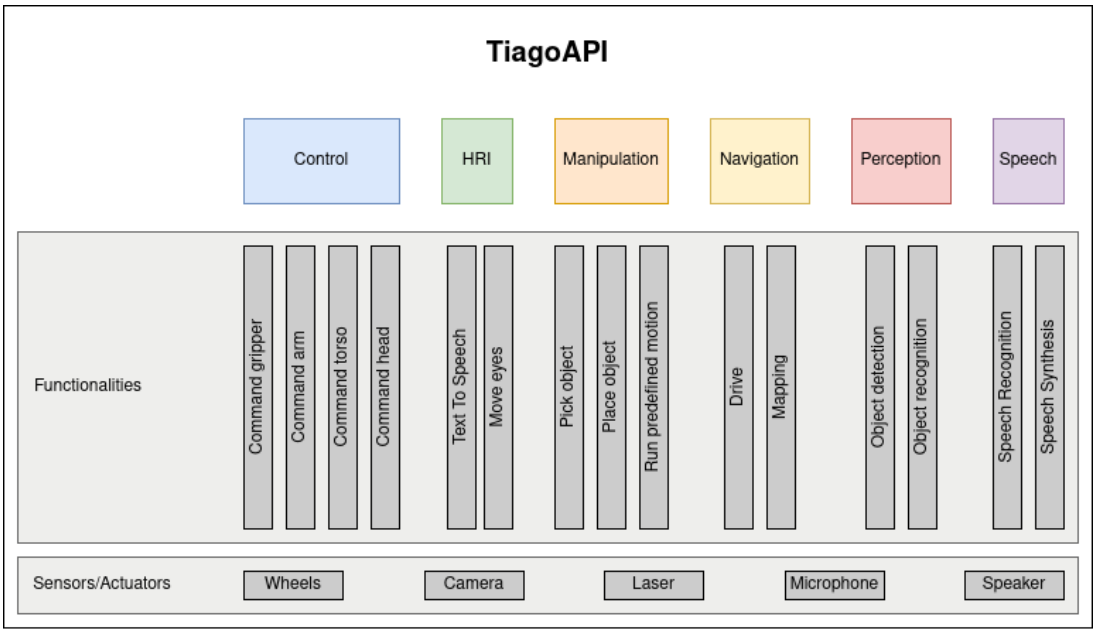

Figure 4.2: TiagoAPI Layered Architecture

portance of making the robot user-friendly and capable of understanding and responding to human commands and inquiries.

- Manipulation: The Manipulation component was created to focus on tasks related to the robot's physical interaction with its environment. This encompasses functionalities such as picking objects, placing objects, and grasp pose estimation. These capabilities are vital for the robot to interact with and manipulate objects in its surroundings accurately and effectively.

- Navigation: Navigation is a fundamental aspect of a robot's autonomy. This component encompasses functionalities related to mapping, localization, and autonomous movement. By developing a sophisticated navigation system, the robot can operate in various environments while avoiding obstacles and reaching specific destinations.

- Perception component is foundational for the robot to actively interpret and respond to its surroundings. Beyond just sensing, it encompasses advanced functionalities like environment modeling, tracking, object detection, and recognizing individuals. These capabilities not only provide the robot with heightened situational awareness but also allow it to make informed decisions based on its environment. The integration of perception and navigation is indispensable for tasks

like person following and object recognition.

- Speech: Speech-related functionalities, including speech recognition and language understanding, are grouped under this component. Effective speech interaction is crucial for enhancing the robot's communication capabilities. The development of a robust dialogue system ensures that the robot can comprehend user commands and engage in meaningful conversations.

4.2.1.C Debugging

The development and improvement of TiagoAPI was intensive and demanding, but it required careful attention to uphold its quality and reliability. To make sure the API is both strong and without flaws, a detailed debugging system was established, bolstered by static code analysis, unit tests and logging tools.

Unit tests, vital to the debugging approach, were set up to cover the entire range of the API. These tests scrutinize individual code segments, like functions, to confirm their accuracy. By separating these parts and testing them against specific inputs, the tests verify that the outcomes match the expected results. Given the continuous evolution of robotic software, these tests ensure that code changes don't accidentally bring about mistakes or setbacks.

Enhancing the feedback from unit tests, a refined logging system was integrated into the API. This records ongoing system activities, events, and modifications. If unexpected issues arise, developers can review these logs to pinpoint the actions leading to the irregularity. With time stamps, background details, and error notes, the logs offer a comprehensive view, making it easier to spot and solve problems.

Beyond just development tools, we recognized the importance of clear documentation for TiagoAPI. Proper documentation not only facilitates developers in understanding the functionalities and intricacies of the API but also serves as a reference guide, ensuring consistent implementation and optimal usage.

To sum it up, debugging for the TiagoAPI was an essential phase in its creation. Thanks to unit tests, logging, simulations, CI tools, and code checks, the API represents top-tier software practices, promising users a reliable and streamlined robotic interaction platform.

4.2.2 Auto Diagnostics Service

In parallel with the development of the API, a diagnostic system was devised, serving as an integral component of the robotic framework. The primary aim of this system, as depicted in Equation 4.1, is to analyze the robot's state during initialization or predefined intervals.

$$\text{Objective(Auto Diagnostics Service)} = \begin{cases} \text{Analyze(State(Robot)) at initialization or} \\ \text{at time intervals, and dynamically enable/} \\ \text{disable components based on outcome} \end{cases} \tag{4.1}$$

Depending on the diagnostic outcome, this system dynamically modifies the active components of the API. This flexibility ensures the API's transferability across different robotic platforms.

The diagnostic system's operation is rooted in the scrutiny of two critical files: Unified Robot Description Format (URDF) and Semantic Robot Description Format (SRDF). These files serve as comprehensive representations of the robot's physical and semantic characteristics. The diagnostic system meticulously inspects these files, examining the presence of essential elements such as nodes, joints, and topics within the robot's architecture.

Consider an illustrative scenario: the absence of a critical arm joint in the URDF or SRDF files. In such an instance, the diagnostic system swiftly identifies this deficiency. Subsequently, it initiates the process of disabling the corresponding component, in this case, the Manipulation component, within the API. This strategic decision ensures that programmers utilizing the API are promptly alerted to the unavailability of the manipulation component. Consequently, they are encouraged to address the issue, either by rectifying the missing joint or by adapting their code to function without the manipulation component's support.

Listing 4.2: Auto Diagnostic Service Overview

```
1 Initialize the robot and diagnostic system.
2 Load URDF and SRDF files for robot description.
```

```
Define a list of required nodes, topics, and joints.
Perform parallel checks on nodes, topics, and joints.
Identify missing or unavailable elements.
If any elements are missing:
    - Disable corresponding API components.
    - Notify the programmer about the unavailability.
Continue robot operation.
Allow dynamic updates to API components if needed.
Loop to periodically re-run diagnostic checks.
```

Listing 4.2 offers a high-level overview of the operational workflow of the robot diagnostic system. It encapsulates the key steps involved, from initialization and file loading to parallel checks and dynamic component management, all in the pursuit of ensuring the adaptability of the API across diverse robotic platforms.

In summary, the integration of this diagnostic system empowers the API with adaptability, allowing it to seamlessly transition from one robot to another. By proactively assessing the robot's health and responding intelligently to its configuration, this diagnostic system plays a pivotal role in ensuring the API's robustness and reliability across diverse robotic platforms.

4.2.3 API CMD Tool

To build upon prior accomplishments, we introduced the API CMD Tool as a bridge, seamlessly connecting developers to the TiagoAPI. This tool provides an interactive platform for engaging with the robot's features. Recognizing the complexities inherent to the ROS ecosystem and the possible barriers to entry for users unfamiliar with the underlying Python infrastructure, the API CMD Tool serves as an abstraction layer that encapsulates the technicalities while offering an intuitive touchpoint for users.

At the heart of this tool is an innovative shell script, `interactive_node.sh`. Its primary purpose is to provide developers with an immediate interactive session for real-time experimentation, leveraging the capabilities of IPython. IPython, distinguished from the standard Python shell, grants users an enhanced interface replete with superior debugging, auto-completion, and easy access to shell commands, making the robot interaction process more intuitive.

Upon invocation, the script ensures the required ROS package, `actions_tiago`, is

accessible. Once validated, it then ascertains the presence of IPython on the system. If IPython is available, the script initializes it, granting the user direct access to the TiagoAPI through an interactive session. This means developers can directly manipulate, test, and observe robot behaviors without diving deep into the codebase or manually initializing various components.

Complementing this shell script is a Python module, `tiago_api_class_node`, which upon execution, initializes a ROS node tailored for interactions via the TiagoAPI. By ensuring this node's availability, the tool guarantees that developers have the necessary infrastructural support to communicate with the robot in real-time through their IPython session.

In essence, the API CMD Tool embodies the project's overarching emphasis on accessibility and user-friendliness. It serves as a testament to the commitment towards lowering barriers to entry, offering both experienced roboticists and novices alike an efficient, straightforward means to interact with the robot. Through such innovations, the tool ensures that the robot's advanced capabilities are within reach, promoting rapid development, testing, and iterative experimentation.

5

Experiments and Results

Contents

To comprehensively evaluate the TiagoAPI's performance, a thorough assessment was conducted through a series of experiments in diverse settings, encompassing simulated environments and real-world scenarios. 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. Notably, the active utilization in the competition facilitated the identification of shortcomings and the implementation of immediate improvements, following a process of adaptation. Furthermore, valuable feedback from peers and comparative analyses with other systems played a crucial role in the ongoing refinement process.

5.1 Simulated Environments

The evaluation of the TiagoAPI comprised a set of crucial tests performed within simulated environments, primarily utilizing Gazebo. Gazebo is an open-source 3D robotics simulation tool that allows the creation of complex scenarios for testing robots virtually. The simulation environment used, acts as a digital twin of an apartment infrastructure designed from the ISRoboNet@Home, which provides the SocRob@Home team an effective platform for evaluating the robot's performance. Figure 5.1 illustrates this simulated environment.

Throughout the testing phase in the simulated environment, the TiagoAPI's capabilities were evaluated based on the successful execution of various ROS tutorials. These included the following key demonstrations:

- Pick & Place Demo: This demonstration entailed a tabletop pick and place task, integrating monocular model-based object reconstruction through the utilization of ArUco, an open-source library for detecting square markers. The pick and place pipeline in MoveIt!, a widely-used motion planning framework for ROS, facilitated the seamless manipulation of objects, highlighting the TiagoAPI's proficiency in precise and coordinated object interaction within the simulated environment.

- Planning with Octomap Demo: This test emphasized the application of Octomap in MoveIt! to calculate collision checks within the robot's surrounding environment

Figure 5.1: ISRoboNet@Home Testbed Simulated

during motion planning in Cartesian space. Octomap is an efficient 3D mapping framework that utilizes octrees for representing the environment. By effectively integrating Octomap, the TiagoAPI demonstrated its competence in navigating intricate environments and planning collision-free trajectories. This demonstration further affirmed its adaptability and reliability in real-world scenarios.

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

5.2 Real-World Scenarios

During the evaluation process of the TiagoAPI, the exploration extended beyond simulated environments, encompassing a diverse array of real-world scenarios. This approach enabled a more holistic understanding of the TiagoAPI's practical applicability and performance in various complex, dynamic, and uncontrolled settings.

To illustrate the experiments and results, a task from the RoboCup competition, the "Receptionist" task, serves as an illustrative example. This task involves proficiently guiding and introducing two new guests to a lively party in the living room while maintaining appropriate gaze direction, thereby ensuring engaging human-robot interac-

tion. This challenge highlights the necessity for seamless system integration, effective human-robot interaction, accurate person detection, and recognition capabilities. The task demands the robot to navigate the living room skillfully, attend to arriving guests, and assist in managing seating arrangements. The detailed state machine, illustrated in Figure 5.2 below, outlines a sequential process involved in successfully completing the 'Receptionist' task during the RoboCup competition.

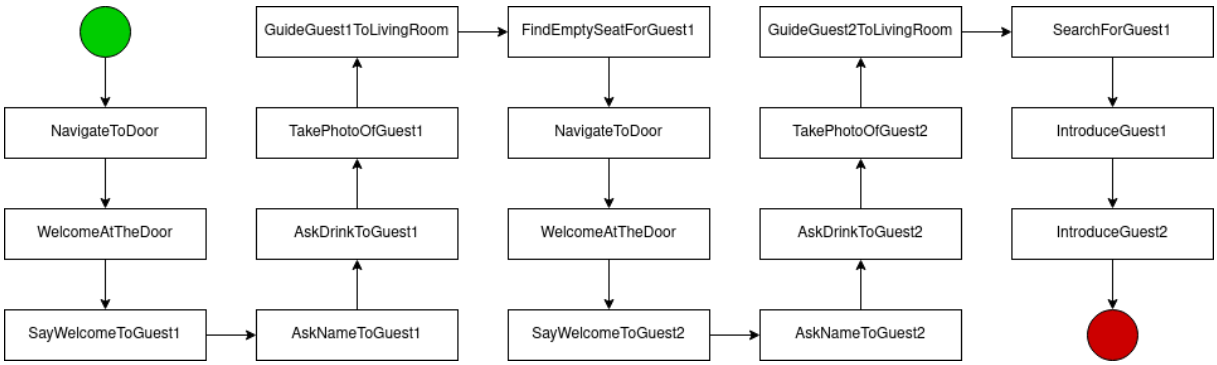

Figure 5.2: RoboCup Task Receptionist - State Machine for Successful Execution

In this context, a state machine serves as a conceptual model illustrating the various operational phases a robot can undertake, and the transitions between these phases in response to specific events or stimuli. A state represents an operational mode assumed by the robot at any given time.

For example, let's consider the state 'FindEmptySeatForGuest1' as an illustrative case. This state necessitates the robot's analysis of whether a particular seat is currently occupied or unoccupied. To accomplish this task, the system effectively leverages the TiagoAPI's perception component.

Remarkably, the majority of the functions employed within the state machine make use of the TiagoAPI, highlighting the seamless integration of the TiagoAPI's capabilities in executing a variety of intended behaviors. This integration significantly contributes to the efficient operation of the robotic system within the predefined states and transitions of the state machine framework.

The 'Receptionist' task performed can be viewed in action through the following link: https://www.youtube.com/watch?v=Q7XkN0Y095o.

6

Conclusions

Contents

In this study, an accessible and user-friendly API was developed for the TIAGo mobile service robot, aiming to streamline its functionalities and simplify its control mechanism. The proposed solution, the TiagoAPI, takes a resource-oriented approach, providing developers with a unified and agnostic platform for interacting with the robot's core functionalities. Through rigorous implementation and integration efforts, the effectiveness of this approach was successfully demonstrated, underscoring its potential to revolutionize the field of robotics.

The solution not only caters to experienced developers but also addresses the needs of individuals from diverse backgrounds, including those without a deep understanding of robotics. By encapsulating and documenting TIAGo's functionalities within a single, comprehensive framework, the barrier to entry has been effectively lowered, enabling users to combine various functions seamlessly for a multitude of tasks.

Moreover, the implementation of the TiagoAPI was complemented by a diagnostic system designed to ensure the transferability of the API across diverse robotic platforms. This system, characterized by its analysis of the robot's configuration files, plays a role in facilitating swift adjustments and dynamic modifications. By proactively assessing the robot's health and responding intelligently to its configuration, this diagnostic system enhances the API's versatility and makes it inherently transferable. The system's ability to adapt to the nuances of different robotic platforms positions the TiagoAPI as a future-proof solution, ready to seamlessly transition and integrate into a wide spectrum of robotic systems.

The integration of the API CMD Tool further solidifies the commitment to accessibility and user-friendliness. By providing an intuitive shell script and a Python module, users can interact with the TiagoAPI effortlessly, fostering an environment conducive to rapid development, testing, and iterative experimentation.

Overall, the research represents a significant step forward in the field of robotics, emphasizing not only the technical intricacies of robot development but also the crucial importance of accessibility, adaptability, and user-friendliness. By paving the way for a more inclusive and intuitive robotic ecosystem, the TiagoAPI serves as a stepping stone towards a future where robotics seamlessly integrates into everyday life, offering innovative solutions to complex challenges and enhancing human-machine interactions.

Moving forward, the TiagoAPI is expected to continue to evolve, incorporating advancements in technology and responding to the changing needs of the robotics community. Further research and development in this direction are encouraged, fostering

collaboration and knowledge-sharing to propel the field of robotics to new heights.

6.1 Future Work

The TiagoAPI, as it stands, offers a robust framework for robotic functionalities. However, like all evolving systems, there are areas that could benefit from further refinement and exploration. One of the main challenges is ensuring the API's ability to have a generalized understanding of diverse tasks. Drawing from the concept of transferable learning systems, there's potential for a system to understand and adapt to a wide range of tasks based on its prior knowledge.

Specifically for TiagoAPI, future enhancements could focus on:

1. Enhanced Transferability Across Diverse Robotic Platforms: While TiagoAPI has shown adaptability with its diagnostic service, there's potential to expand this capability to ensure seamless integration with an even broader range of robotic platforms. By investing in this area, the goal would be to make the API universally compatible, reducing the need for custom adaptations and configurations when transitioning between different robot types and models.

2. Improved Diagnostics and Feedback: While the API offers some level of diagnostics, further enhancements in real-time feedback and error correction can make the user experience even smoother. For instance, if a command doesn't execute as expected, instead of merely indicating an error, the system could elucidate the underlying reason—be it a hardware malfunction, a software glitch, or an environmental factor. By offering a more granular understanding of the robot's operational status, users can efficiently troubleshoot and refine their interaction with the system, leading to more robust and reliable applications.

3. Intuitive User Interface and Experience: As the field of robotics continues to grow and attract a diverse range of users, from seasoned developers to novices, the need for an intuitive interface becomes paramount. Future iterations of the TiagoAPI could incorporate a more user-friendly interface that caters to both expert and beginner users. This could involve visual programming tools, drag-and-drop functionalities, or context-aware help systems that guide users through the intricacies of robot programming, making the process more accessible and streamlined.

In conclusion, while the current TiagoAPI has made significant strides in simplifying robot programming, future versions have the potential to offer even more sophisticated, yet user-friendly functionalities. By addressing the aforementioned gaps and building on the existing framework, the TiagoAPI can become an even more powerful tool for roboticists and developers.

Bibliography

[1] 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*, vol. 78, no. 6-13, 2016.

[2] 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*, vol. 324, no. 5923, pp. 85–89, 2009.

[3] P. Lin, G. Bekey, and K. Abney, "Autonomous military robotics: Risk, ethics, and design," California Polytechnic State Univ San Luis Obispo, Tech. Rep., 2008.

[4] 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*, vol. 1, no. 2, pp. 113–118, 2007.

[5] T. Belpaeme, J. Kennedy, A. Ramachandran, B. Scassellati, and F. Tanaka, "Social robots for education: A review," *Science robotics*, vol. 3, no. 21, p. eaat5954, 2018.

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

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

[8] M. Fink, H. Gassen, and U. Reiser, "Domestic service robots: A market overview," *International Journal of Advanced Robotic Systems*, vol. 7, no. 4, pp. 365–378, 2010.

[9] J. Lee, T.-W. Kim, and S.-H. Kim, "Domestic robots: Status and prospects," *International Journal of Advanced Robotic Systems*, vol. 9, no. 4, pp. 1–15, 2012.

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

[11] 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*, vol. 33, no. 4, pp. 343–356, 2019.

[12] "Equipa do técnico alcança 2.º lugar em competição internacional de robótica," https://tecnico.ulisboa.pt/pt/noticias/equipa-do-tecnico-alcanca-2-o-lugar-em-competicao-internacional-de-robotica/, accessed: 2023-09-27.

[13] S. Kounev, T. Hildenbrand, and F. Leymann, "A survey of middleware for the internet of things," *IEEE Internet Computing*, vol. 15, no. 6, pp. 50–57, 2011.

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

[15] 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*, vol. 8, pp. 162 315–162 325, 2020.

[16] S. Khan, M. Z. Qureshi, N. Ahmad, and L. Zaman, "A survey of robotic apis: Design and implementation," *IEEE Access*, vol. 7, pp. 124 596–124 613, 2019.

[17] G. Van Rossum, "An overview of the python language," *Computer*, vol. 23, no. 11, pp. 31–41, 1990.

[18] F. Perez and B. E. Granger, "Ipython: a system for interactive scientific computing," *Computing in Science and Engineering*, vol. 9, no. 3, pp. 21–29, 2007.

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

[20] M. Quigley, K. Conley, B. Gerkey *et al.*, "Ros: An open-source robot operating system," *ICRA Workshop on Open Source Software*, vol. 3, no. 3.2, p. 5, 2009.

[21] 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*, vol. 16, no. 6, p. 1729881419864261, 2019.

[22] 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)*. IEEE, 2014, pp. 3376–3383.

[23] LinkedIn. (2023) Pros and cons of using ros (robot operating system) in robotics. Accessed on September 30, 2023. [Online]. Available: https://www.linkedin.com/advice/3/what-pros-cons-using-ros-robot-operating-system-robotics

[24] 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*, 2010, pp. 4036–4041.

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

[26] G. Metta, P. Fitzpatrick, and L. Natale, “Yarp: Yet another robot platform,” *International Journal of Advanced Robotic Systems*, vol. 3, no. 1, p. 043, 2006.

[27] P. Fitzpatrick, G. Metta, and L. Natale, “Towards long-lived robot genes,” *Robotics and Autonomous Systems*, vol. 56, no. 1, pp. 29–45, 2008.

[28] 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*, vol. 3, p. 26, 2016.

[29] M. Jackson, “Microsoft robotics studio: A technical introduction,” *IEEE Robotics Automation Magazine*, vol. 14, no. 4, pp. 82–87, 2007.

[30] A. Burke and J. Lasenby, “Using the microsoft robotics developer studio as a simulation tool for multi-robot systems,” *International Journal of Computer Applications*, vol. 29, no. 12, pp. 22–28, 2011.

[31] G. Hu, W. P. Tay, and Y. Wen, “Cloud robotics: architecture, challenges, and applications,” *IEEE Network*, vol. 26, no. 3, pp. 21–28, 2010.

[32] A. Rosenfeld and B. Green, “Orocos: An in-depth analysis of a modular robot control framework,” *Journal of Robotic Systems*, vol. 39, no. 4, pp. 321–334, 2022.

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

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

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