Intro to Robot Operating System

by Siddharth Patel

vol.1

In memory
of
Ashokbhai Babubhai Patel

Intro to Robot Operating System - Using ROS in Python

Learn to program Robots using the famous Robot Operating System (ROS) framework in Python

Siddharth Patel

September 28, 2024

Preface

One vivid memory from my childhood is the time when LED TVs were gaining popularity in India. I constantly wondered how the images were displayed and what those tiny, intricate components—like miniature buildings—were doing inside. I soon realized that the failure of even one small part could stop the entire system from working. This taught me an important lesson: there are countless ways for a circuit to fail, but only a few precise combinations that allow it to function flawlessly. That insight stayed with me and shaped my approach to problem-solving in electronics.

Through these early experiences, I came to a deeper realization about how computers, electronics, and even machines like water pipes all share something fundamental: they behave like laborers—obedient, yet extremely limited laborers. Why do I call them ”super-dumb”? Because they only follow instructions without question or deviation. Take, for example, a green LED and a blue LED. Neither knows how to collaborate or blend their light to create a cyan color on their own. They simply light up when the correct voltage is applied. The intelligence to make them work together, to adjust their brightness and achieve the desired color, lies with us. We design the circuit, we provide the instructions, and it's through our guidance that these ”laborers” fulfill their purpose.

I applied this same principle early on, combining different circuits and components to create devices with new, unique functions. This hands-on experimentation not only fueled my creativity but also expanded my knowledge of circuits, integrated circuits (ICs), sensors, and eventually coding. However, in 2023, I discovered something that took my understanding to the next level—a development by two visionary minds, Keenan Wyrobek and Eric Berger. These two pioneers, who shared my passion for robotics and scientific innovation, were far ahead of me in their realization of just how challenging robotics development can be. With so many components at play, countless things can go wrong, and only a few will function as intended. They referred to this ongoing struggle as ”reinventing the wheel”—a concept that resonated deeply with my own experiences in building and troubleshooting complex systems.

Imagine if, every time you made a pizza, you had to buy a farm, grow the grains, make the dough, and ferment the cheese yourself. It would take forever! Instead, we buy ingredients from the store and make the pizza. Wyrobek and Berger applied the same logic to robotics by creating ROS (Robot Operating System), which allows different parts (or ”laborers”) to communicate with each other. The best part is that you don’t need to reinvent the wheel—someone else has already created it, and you can focus on putting the parts together to build complex systems or robots.

In 2023, I started working with my professor, Prof. Dr.-Ing. Jan Hanno Carstens, on developing a robot for his company. I was forced to learn ROS because we were using it for our project. It took me a long time, partly because I found the ROS documentation a bit overwhelming, and I wasn’t sure where to start. So, during my vacation, I decided to document everything I had learned in this book, using one of my favorite approaches to learning.

Let me explain my approach with a great example that I used in another area of my life. When I came to Germany, I had to learn the German language to study at a prestigious German university. Learning a language was a big challenge for me because my mind works logically, If something doesn’t make sense, I find it hard to learn. Languages don’t have formulas you can memorize and then pass an exam the next day. You need to invest time every day for at least a year or two to become fluent.

But I didn’t have that kind of time, so I developed a strategy. I realized that while German may have thousands of verbs, only a few hundred are used in daily conversation, especially in an engineering context. By focusing on these 200–300 verbs, I was able to learn the language quickly. This approach worked—I earned a scholarship, got a job at my university, and competed successfully with native German speakers.

That’s exactly what I’m doing with this book. I’ve compiled all the most common ROS commands and concepts you’ll need to get started quickly. This book is for people who want to learn ROS but don’t have a year to study—they have a job starting next week! By going through this book, they can probably manage just fine.

In my next book, I plan to present 10 ROS projects that I created, ranging from simple to complex, to show you how to apply ROS in real-world scenarios.

Abstract

This book is designed to offer an accelerated yet comprehensive introduction to the Robot Operating System (ROS), aimed at individuals who, like myself, may not have the luxury of years to master a complex system. Whether you're a researcher, engineer, or hobbyist, ROS provides a powerful framework that allows you to focus on innovation without constantly ”reinventing the wheel.”

ROS emerged from a fundamental need to streamline the development of robotics systems—a challenge that has resonated with roboticists for years. In this book, I trace the origins of ROS, from its creation by Keenan Wyrobek and Eric Berger as a solution to the inefficiency of duplicating low-level code for every new robotics project, to its present status as a global standard in the field. The opening chapters explain how ROS was born from a desire to enable developers to focus on building intelligent algorithms, rather than repeatedly coding the same infrastructure from scratch.

At its core, ROS is a middleware—a system that sits between the robot's hardware and the software, providing modular tools that allow components to communicate seamlessly. The book introduces key concepts like nodes, topics, services, and actions, explaining how each element plays a role in building complex systems with reusable, scalable, and efficient components. By drawing on these building blocks, you'll learn how to write robotics software that's both powerful and adaptable, without needing to start from scratch for every project.

After introducing the ROS framework, the book walks you through setting up your ROS development environment, using step-by-step instructions that focus on simplicity and clarity. You'll learn how to install ROS Noetic, create workspaces, and develop your first ROS publisher and subscriber scripts. These initial projects will get you up and running quickly, providing practical experience in managing nodes, publishing data to topics, and processing messages in real-time.

Moving beyond the basics, the book delves into more advanced concepts such as ROS services and actions, parameter servers, and how to record and play back data using ROS bag files. Each concept is broken down with clear explanations and practical examples, ensuring that you

can apply these tools in real-world scenarios.

Throughout the book, my goal is to minimize the steep learning curve that often comes with ROS. Drawing from my own experience, I've compiled a list of essential commands, functions, and best practices that will help you navigate the ROS ecosystem efficiently. The final chapters explore how to use launch files to simplify the process of running multiple nodes, and how to manage your workspace in a way that scales as your projects grow more complex.

With the help of this book, you'll not only grasp the fundamentals of ROS but also gain the confidence to start applying your knowledge immediately. In much the same way I approached learning German by focusing on its most essential elements, this book offers a targeted approach to learning ROS—distilling the vast and often complex documentation into the core concepts and commands that will get you up and running quickly.

For those eager to move beyond the basics, stay tuned for my next book, where I will dive into 10 real-world ROS projects—demonstrating how to build everything from simple robots to fully autonomous systems using the power of ROS.

Contents

List of Figures

What is ROS?

1

1.1 History of ROS

ROS (Robot Operating System) has become a standard in the field of robotics, widely adopted by researchers, hobbyists, and even large robotics companies. But the path to its current success was not straightforward. The history of ROS goes back to the mid-2000s, and its beginnings were humble, born out of the need to solve a common problem in robotics.

1.1.1 The Stanford Period

ROS started as a personal project of Keenan Wyrobek and Eric Berger while they were at Stanford University. During that time, robotics development faced a serious challenge: developers had to spend too much time reinventing basic software infrastructure, such as sensor drivers and actuator controllers, for every new project. As a result, there was little time left for developing advanced robotic intelligence.

Even within the same organizations, developers would re-implement these core systems for each project. This situation was frustrating for robotics engineers, including Keenan and Eric, who wanted to focus on developing complex robotic algorithms instead of constantly rebuilding the wheel.

What does ”Reinventing the Wheel” Mean?

”Reinventing the wheel” refers to the unnecessary duplication of work by recreating something that already exists. In the case of robotics, this meant writing the same code for basic robot functionality over and over again, instead of building upon previous work.

Analogy: Reinventing the Wheel

Imagine your father plants a tree and waters it every day for his entire life. By the time he's 60, the tree is large and bears delicious fruit. However, instead of continuing to nurture this tree, you decide to plant your own tree from scratch. This new tree takes years to grow, while your father's tree could have been even more fruitful with a little extra care.
This is the essence of what ROS aimed to solve. Instead of each robotics project starting from scratch, developers could build upon the infrastructure created by others, allowing them to focus on creating smarter, more complex robots.

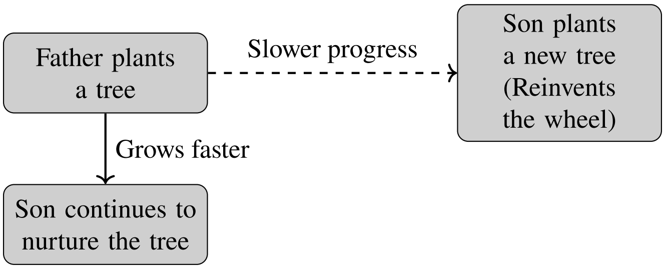
Father plants a tree
Slower progress
Son plants a new tree (Reinvents the wheel)
Grows faster
Son continues to nurture the tree

Figure 1.1: Analogy of Reinventing the Wheel vs. Building on Existing Work

In 2006, Keenan and Eric founded the **Stanford Personal Robotics Program** to address this issue. Their goal was to create a framework that allowed different processes (nodes) to communicate with each other and provide tools to help build code on top of this foundation. The testbed for this framework was a robot they built, known as the *Personal Robot*. They created 10 of these robots and distributed them to universities for development and testing purposes.

1.1.2 Building the Foundation

The Stanford Personal Robotics Program laid the foundation for ROS. The key idea was to have a common framework where developers could share software components, such as drivers and communication systems. This would allow roboticists to focus on creating smarter, more innovative applications rather than rewriting basic infrastructure. ROS was designed to scale and adapt to different robots and environments.

The Foundation of ROS

The Stanford Personal Robotics Program aimed to create a universal framework for robot development. This framework allowed different robot processes to communicate seamlessly, and it came with tools to build code on top of the existing infrastructure. The robot built for this program, the *Personal Robot*, was distributed to universities to promote ROS development.

The idea was simple but revolutionary: let roboticists collaborate and build on top of each other's work. Much like the analogy of the father's tree, ROS would allow developers to nurture and grow from the work already done by others, avoiding the pitfalls of reinvention.

The development of ROS during the Stanford Period is well documented. The vision was clearly laid out in a fundraising deck from 2006, which highlighted the need to stop reinventing the wheel and instead build a reusable framework for robotics software.

1.2 Challenges in Robotics Development

Building robots is inherently complex due to the wide range of hardware and software involved. Historically, roboticists often faced the challenge of writing every piece of the operating code from scratch, typically in low-level languages like C. Even simple tasks, such as making a light blink, could take weeks of effort due to the numerous components involved.

Example: Imagine trying to code a blinking light indicator for your robot. You would need to:

Challenges in Coding a Blinking Light Indicator

- **Create the firmware controller:** Write the low-level code to manage the microcontroller.
- **Manage serial communications:** Ensure the microcontroller can communicate with a host computer.
- **Create higher-level software nodes:** Build logic to decide when the light should blink.
- **Develop debugging and visualization tools:** Ensure all the sensors, actuators, and code are functioning correctly.

This entire process represents a significant time and effort investment for what should be a simple task.

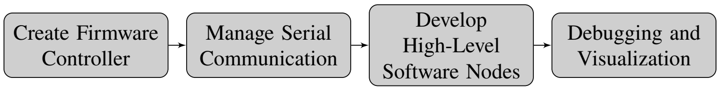
Create Firmware Controller
Manage Serial Communication
Develop High-Level Software Nodes
Debugging and Visualization

Figure 1.2: Robotic Development Workflow: Even a simple task like a blinking light requires numerous steps.

The complexity doesn't stop here. For each new project or robot, engineers often had to repeat these steps. This process of "reinventing the wheel" for every project significantly slowed down the pace of robotics development, which led to the birth of ROS.

1.3 The Birth of ROS

The challenge of re-implementing basic robotic functionality for every project led to the creation of ROS (Robot Operating System). Born as an **open-source middleware**, ROS provides a set of standardized tools and libraries that drastically reduce the time needed to develop robotic systems.

Initially developed at Stanford University in 2006 as part of the **Stanford Personal Robotics Program**, ROS was created with the goal of offering reusable software infrastructure, enabling researchers and developers to focus on building intelligent systems, instead of wasting time on reimplementing low-level drivers and communication protocols.

ROS simplifies robotic development by providing:

Key Benefits of ROS

- **Open-source:** Freely accessible, allowing for global collaboration.
- **Reduced development time:** Built-in tools, drivers, and libraries for common robotics tasks.
- **Modular design:** Promotes reusable components and allows for easy system scaling.
- **Platform independence:** Initially developed for Linux, ROS now has experimental support for macOS and Windows.

1.4 What ROS Is and Isn't

ROS is often misunderstood as a traditional operating system, but it is more accurately described as a **meta-operating system** or **middleware**.

Key Benefits of ROS is:

- An **open-source middleware** that sits on top of a traditional operating system (like Ubuntu).
- A **development environment** with tools for building robotic systems, including visualization (Rviz), communication libraries, and introspection tools.
- A **packaging system** that supports the distribution of robot software in reusable modules. ROS uses the **colcon** command to build and manage these packages.

ROS is not:

- A **computer operating system**. It is not a replacement for Linux, macOS, or Windows. It runs on top of these OSs.
- A **programming language**. ROS programs are written in languages like C++ and Python. Other experimental languages include Java, Lisp, and Octave.
- A **hard real-time environment**. ROS is not suitable for systems requiring hard real-time constraints.
- A **development environment**. ROS is used with existing IDEs or text editors like Sublime or VSCode.

1.5 Why Learn ROS?

ROS has emerged as the standard framework for developing robotic systems due to several key advantages.

1.5.1 Open Source

ROS is entirely **open-source**, which means the community is constantly contributing to and improving the software. The collaborative nature of ROS allows developers to share their work, learn from each other, and continuously improve robotic development worldwide.

1.5.2 Reusability and Modularity

With ROS, you don't have to start from scratch every time. The framework provides numerous **open-sourced tools** and libraries, which allow developers to easily contribute, adapt, and share software.

1.5.3 Support for Development Tools

ROS comes with built-in tools to assist with development. For example:

Development Tools Examples:

- **Rviz:** A 2D/3D visualization tool for representing robot data like sensor information and environments.
- **Gazebo:** A simulation tool that allows you to test robotic systems in a virtual environment before deploying on actual hardware.

1.5.4 Rapid Testing and Prototyping

ROS provides a platform where you can quickly prototype robotic systems using simulators like Gazebo or test data using bag files (rosbags). This allows you to refine your system design before deploying it to the physical robot.

1.6 Languages Supported by ROS

ROS natively supports several programming languages, providing flexibility to developers based on their needs.

1.6.1 Officially Supported Languages

ROS is primarily used with:

Officially Supported Languages

- **C++:** Often used for performance-critical tasks in robotics due to its speed and control over hardware.
- **Python:** Highly popular for writing simple and quick scripts, offering ease of use and rapid development.
- **Lisp:** Historically supported but less commonly used in modern ROS applications.

1.6.2 Community-Driven Libraries

ROS also has community-driven support for additional languages, enabling greater flexibility for developers.

Community-Driven Libraries

- **Java:** Enables the integration of robotics into Java-based systems.
- **JavaScript:** Can be used in web-based applications for robotics control.
- **Lua:** A lightweight scripting language useful for specific robotic tasks.

1.7 Conclusion

With the vast amount of existing ROS 1 libraries and new features in ROS 2, roboticists can leverage these tools to develop powerful and scalable robotic systems. ROS is already a major player in the robotics field, and learning it will significantly enhance your ability to create and deploy robotics solutions efficiently.

Let's dive deeper into ROS and explore its capabilities!