

# Intro to Robot Operating System

by Siddharth Patel



Hochschule für Technik  
und Wirtschaft Berlin  
University of Applied Sciences

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

<b>1</b>	<b>What is ROS?</b>	<b>19</b>
1.1	History of ROS . . . . .	20
1.1.1	The Stanford Period . . . . .	20
1.1.2	Building the Foundation . . . . .	21
1.2	Challenges in Robotics Development . . . . .	22
1.3	The Birth of ROS . . . . .	23
1.4	What ROS Is and Isn't . . . . .	24
1.5	Why Learn ROS? . . . . .	25
1.5.1	Open Source . . . . .	25
1.5.2	Reusability and Modularity . . . . .	25
1.5.3	Support for Development Tools . . . . .	25
1.5.4	Rapid Testing and Prototyping . . . . .	26
1.6	Languages Supported by ROS . . . . .	26
1.6.1	Officially Supported Languages . . . . .	26
1.6.2	Community-Driven Libraries . . . . .	27
1.7	Conclusion . . . . .	27
<b>2</b>	<b>Environment Setup</b>	<b>29</b>
2.1	Introduction . . . . .	30
2.2	Installing Ubuntu . . . . .	30
2.2.1	Windows Setup . . . . .	30
2.2.2	macOS Setup . . . . .	30
2.3	Other Installation Methods . . . . .	31
2.4	Installing a Code Editor . . . . .	31
2.5	Conclusion . . . . .	32
<b>3</b>	<b>Installing ROS Noetic</b>	<b>33</b>
3.1	Introduction . . . . .	34

## CONTENTS

---

3.2	Installation	34
3.3	Conclusion	37
<b>4</b>	<b>ROS Framework Overview</b>	<b>39</b>
4.1	Introduction	40
4.2	ROS Master	40
4.3	Nodes	40
4.3.1	How Nodes Communicate	41
4.3.2	Node Lifecycle	43
4.4	Topics	43
4.4.1	How Topics Work	44
4.4.2	Example of Topics in Action	44
4.4.3	Node vs Topic: Key Differences	45
4.4.4	Analogy: Nodes and Topics	46
4.5	Services	46
4.6	Actions	47
4.7	Parameter Server	48
4.8	Bag Files	49
4.9	Packages	49
4.10	Conclusion	50
<b>5</b>	<b>ROS Workspace Setup</b>	<b>51</b>
5.1	Introduction	52
5.2	Understanding ROS Workspace Structure	52
5.3	Visualizing Workspace Structure	53
5.4	Creating the Workspace in Ubuntu	53
5.4.1	Step 1: Create Workspace Directory	53
5.4.2	Step 2: Create a ROS Package	54
5.4.3	Step 3: Create Scripts Folder	54
5.5	Compiling the Workspace	54
5.6	Final Workspace Structure	55
5.7	Recap	55
<b>6</b>	<b>Coding a ROS Publisher</b>	<b>57</b>
6.1	Introduction	58
6.2	Writing the Publisher Script	58
6.2.1	Importing ROS Libraries	58
6.2.2	Creating the Main Function	58
6.2.3	Initializing the ROS Node	59

6.2.4	Creating a ROS Publisher . . . . .	60
6.2.5	Setting the Publish Rate . . . . .	60
6.2.6	Publishing the Message . . . . .	61
6.2.7	Importing the Required Message Type . . . . .	61
6.3	Summary Table of Functions . . . . .	62
6.4	Verifying the Publisher with ROS Commands . . . . .	62
6.4.1	Listing Active Nodes . . . . .	62
6.4.2	Listing Topics . . . . .	63
6.4.3	Echoing Topic Messages . . . . .	63
6.4.4	Getting Topic Information . . . . .	64
6.4.5	Monitoring Publish Rate . . . . .	64
6.4.6	Stopping the Publisher . . . . .	65
6.5	Summary Table of ROS Commands . . . . .	66
<b>7</b>	<b>Coding a ROS Subscriber</b>	<b>67</b>
7.1	Introduction . . . . .	68
7.2	Writing the Subscriber Script . . . . .	68
7.2.1	Importing ROS Libraries . . . . .	68
7.2.2	Defining the Main Function . . . . .	69
7.2.3	Initializing the ROS Node . . . . .	69
7.2.4	Creating a ROS Subscriber . . . . .	69
7.2.5	Defining the Callback Function . . . . .	70
7.3	Summary Table of ROS Subscriber Functions . . . . .	71
<b>8</b>	<b>ROS Message Types</b>	<b>73</b>
8.1	Introduction . . . . .	74
8.2	Overview of ROS Message Types . . . . .	74
8.2.1	Standard Message Types . . . . .	74
8.2.2	Custom Message Types . . . . .	75
8.3	Using Sensor Message Types . . . . .	75
8.3.1	Example: Publishing LaserScan Data . . . . .	75
8.4	Using Geometry Message Types . . . . .	76
8.4.1	Example: Using Vector3 Message . . . . .	76
8.5	4. Summary Table of Functions . . . . .	78
8.6	Conclusion . . . . .	78
<b>9</b>	<b>Your First Project in ROS</b>	<b>79</b>
9.1	Introduction . . . . .	80
9.2	Project Overview . . . . .	80

## CONTENTS

---

9.2.1	Mathematical Concept: Transformation Matrix . . . . .	80
9.3	Creating the ROS Package and Workspace . . . . .	81
9.4	Project Breakdown . . . . .	83
<b>10</b>	<b>Solution to Your First Project</b>	<b>85</b>
10.1	Introduction . . . . .	86
10.2	Writing the Publisher Script . . . . .	86
10.2.1	Importing Required Libraries . . . . .	86
10.2.2	Initializing the Publisher Node . . . . .	86
10.2.3	Publishing Random Velocities . . . . .	87
10.2.4	Complete Publisher Script . . . . .	88
10.3	Writing the Subscriber Script . . . . .	89
10.3.1	Importing Required Libraries . . . . .	89
10.3.2	Transformation Matrix and Velocity Calculations . . . . .	90
10.3.3	Defining the Subscriber Callback Function . . . . .	92
10.3.4	Complete Subscriber Script . . . . .	93
10.4	Conclusion . . . . .	95
<b>11</b>	<b>ROS Parameters</b>	<b>97</b>
11.1	Introduction . . . . .	98
11.2	Why Use the ROS Parameter Server? . . . . .	98
11.3	Interacting with the Parameter Server from the Terminal . . . . .	98
11.3.1	Listing Parameters . . . . .	98
11.3.2	Setting and Getting Parameters . . . . .	99
11.4	Saving and Loading Parameters . . . . .	99
11.4.1	Dumping Parameters . . . . .	100
11.4.2	Loading Parameters . . . . .	100
11.4.3	When to Use Dump and Load . . . . .	100
11.5	Integrating ROS Parameters into Python Code . . . . .	101
11.5.1	Using <code>rospy.get_param</code> in <code>subscriber_script.py</code>	101
11.6	Summary Table of ROS Parameter Functions . . . . .	103
<b>12</b>	<b>ROS Basics</b>	<b>105</b>
12.1	Introduction . . . . .	106
12.2	Sourcing a Workspace . . . . .	106
12.2.1	The Need for Sourcing . . . . .	106
12.2.2	Making Sourcing Automatic . . . . .	107
12.3	Running ROS Nodes with <code>roslaunch</code> . . . . .	108
12.3.1	Why Tab Completion May Not Work . . . . .	108

## CONTENTS

---

12.4	Making Python Scripts Executable . . . . .	109
12.4.1	Python Version Error . . . . .	109
12.5	Dealing with Python Version Errors . . . . .	110
12.5.1	Example of Updating a Script . . . . .	110
12.6	Starting roscore in a Parallel Terminal . . . . .	111
12.7	Summary . . . . .	112
12.8	Conclusion . . . . .	112
<b>13</b>	<b>Using Launch Files in ROS</b>	<b>113</b>
13.1	Introduction . . . . .	114
13.2	Creating the Launch Folder . . . . .	114
13.3	Where to Place the Launch File . . . . .	114
13.4	Understanding the Structure of a Launch File . . . . .	115
13.4.1	Tags in Launch Files . . . . .	116
13.5	Creating the Launch File . . . . .	116
13.6	Modifying the Launch File . . . . .	116
13.7	Running the Launch File . . . . .	117
13.8	Conclusion . . . . .	118
<b>14</b>	<b>Your Second Project</b>	<b>119</b>
14.1	Project Overview . . . . .	120
14.2	Visualizing the Project Workflow . . . . .	120
14.3	Project Requirements . . . . .	121
14.4	Things to Take Care of While Doing This Project . . . . .	122
14.4.1	Sourcing Your ROS Workspace . . . . .	122
14.4.2	Making Python Scripts Executable . . . . .	122
14.4.3	Specifying Python 3 in Your Scripts . . . . .	122
14.4.4	Starting roscore in a Parallel Terminal . . . . .	123
14.5	Conclusion . . . . .	123
<b>15</b>	<b>Solution to Second Project</b>	<b>125</b>
15.1	Introduction . . . . .	126
15.2	Creating the Launch File . . . . .	126
15.2.1	Basic Structure of the Launch File . . . . .	126
15.2.2	Setting Parameters . . . . .	127
15.2.3	Launching the Publisher Node . . . . .	127
15.2.4	Launching the Subscriber Node . . . . .	128
15.2.5	Complete Launch File . . . . .	128
15.3	Things to Keep in Mind [14.4] . . . . .	129

15.4	Testing the Launch File . . . . .	129
15.5	Summary of Key Commands and Concepts . . . . .	130
15.6	Conclusion . . . . .	130
<b>16</b>	<b>ROS Bag Files</b>	<b>131</b>
16.1	Introduction . . . . .	132
16.2	What are ROS Bag Files? . . . . .	132
16.3	Project Overview: Temperature Sensor Simulation . . . . .	132
16.3.1	Things to Keep in Mind . . . . .	132
16.4	Writing the Publisher Script . . . . .	133
16.4.1	Importing Required Libraries . . . . .	133
16.4.2	Initializing the Publisher Node . . . . .	134
16.4.3	Publishing Temperature Values . . . . .	134
16.4.4	Complete Publisher Script . . . . .	135
16.5	Writing the Subscriber Script . . . . .	136
16.5.1	Importing Required Libraries . . . . .	136
16.5.2	Defining the Callback Function . . . . .	137
16.5.3	Complete Subscriber Script . . . . .	138
16.6	Recording a ROS Bag File . . . . .	138
16.6.1	What Does it Mean to Record a Bag File? . . . . .	138
16.6.2	Recording the Temperature Status Topic . . . . .	139
16.7	Playing a ROS Bag File . . . . .	140
16.7.1	What Does it Mean to Play a Bag File? . . . . .	140
16.7.2	Playing the Test Bag File . . . . .	140
16.7.3	Modifying Playback Speed . . . . .	141
16.7.4	Inspecting the Bag File . . . . .	142
16.8	Summary Table of ROS Bag Commands . . . . .	142
16.9	Conclusion . . . . .	142
<b>17</b>	<b>Exploring ROS Packages</b>	<b>145</b>
17.1	Introduction . . . . .	146
17.2	Why Use ROS Packages? . . . . .	146
17.3	Installing the USB Camera Package . . . . .	146
17.3.1	Installing the Package via <code>sudo apt-get install</code> . . . . .	146
17.4	Verifying the Installation . . . . .	147
17.4.1	Using <code>roscd</code> . . . . .	147
17.4.2	Using <code>rospack list-names</code> . . . . .	147
17.5	Running the USB Camera Node . . . . .	148
17.5.1	Starting <code>roscore</code> . . . . .	148



## CONTENTS

---

17.5.2	Running the <code>usb_cam</code> Node	148
17.6	Viewing Camera Output in ROS	149
17.6.1	Checking Available Topics	149
17.6.2	Echoing the Camera Data	150
17.7	Visualizing the Camera Output in RViz	150
17.7.1	Starting RViz	151
17.8	Using Launch Files for the USB Camera	151
17.8.1	Running the Launch File	151
17.9	Troubleshooting USB Camera Issues	152
17.10	Summary of Commands	153
17.11	Conclusion	153
<b>18</b>	<b>Services in ROS</b>	<b>155</b>
18.1	Introduction to Services in ROS	156
18.2	What is a Service in ROS?	156
18.3	How Do Services Differ from Topics?	156
18.4	Why Are Services Useful?	157
18.5	ROS Services vs Other Programming Tools	158
18.6	Diagram: Client-Server Communication in ROS Services	159
18.7	Creating a Simple Service: Sum of Two Numbers	159
18.7.1	Project Overview	159
18.7.2	Creating the Service Definition	160
18.7.3	Setting Up the ROS Package	160
18.7.4	Creating the Service Definition File	161
18.7.5	Modifying the <code>CMakeLists.txt</code> and <code>package.xml</code> Files	162
18.7.6	Building the Package	164
18.8	Writing the Python Service Scripts	164
18.8.1	Writing the Service Server Script	165
18.8.2	Writing the Service Client Script	167
18.8.3	File Organization and Structure	171
18.9	Testing the Service	172
18.9.1	Things to Keep in Mind	172
18.9.2	Running the Service Server	173
18.9.3	Verifying the Service	173
18.9.4	Checking Service Type	174
18.9.5	Inspecting the Service Definition	174
18.9.6	Calling the Service Manually	174

18.9.7	Running the Client . . . . .	175
18.9.8	Summary of Service Commands and Functions . . . . .	175
18.10	Conclusion . . . . .	176
<b>19</b>	<b>Actions in ROS</b>	<b>177</b>
19.1	Introduction . . . . .	178
19.2	What Are Actions in ROS? . . . . .	178
19.2.1	Basic Flow of Actions . . . . .	179
19.3	Comparing Actions with Services and Topics . . . . .	179
19.3.1	Topics . . . . .	179
19.3.2	Services . . . . .	180
19.3.3	Actions . . . . .	180
19.4	Why Are Actions Important? . . . . .	181
19.5	Project Overview: Robot Navigation Using Actions . . . . .	182
19.5.1	Conceptual Approach . . . . .	182
19.5.2	Defining the Action File . . . . .	183
19.5.3	Modifying the Package Configuration . . . . .	184
19.5.4	Things to Keep in Mind . . . . .	186
19.5.5	File Structure . . . . .	186
19.6	Writing the Action Server Script . . . . .	188
19.6.1	Importing Required Libraries . . . . .	188
19.6.2	Initializing the Action Server and Subscribers . . . . .	189
19.6.3	Defining the Callback Function for Action Goal . . . . .	189
19.6.4	Updating the Robot's Position . . . . .	190
19.6.5	Running the Action Server . . . . .	191
19.6.6	Complete Action Server Script . . . . .	191
19.6.7	Summary of New Functions and Commands . . . . .	192
19.7	Writing the Python Action Client Script . . . . .	193
19.7.1	Importing Required Libraries . . . . .	193
19.7.2	Defining the Feedback Callback Function . . . . .	194
19.7.3	Defining the Action Client Function . . . . .	194
19.7.4	Running the Client Script . . . . .	196
19.7.5	Complete Action Client Script . . . . .	198
19.7.6	Summary of New Functions and Commands . . . . .	199
19.8	Writing the Robot State Publisher Script . . . . .	199
19.8.1	Importing Required Libraries . . . . .	199
19.8.2	Defining the Publisher Function . . . . .	200
19.8.3	Running the Publisher Script . . . . .	201

## CONTENTS

---

19.8.4	Complete Robot State Publisher Script . . . . .	202
19.8.5	Summary of New Functions and Commands . . .	204
19.9	Running the Action Nodes and Testing the System . . . .	204
19.9.1	Starting the Action Server . . . . .	204
19.9.2	Starting the Robot State Publisher . . . . .	205
19.9.3	Running the Action Client . . . . .	205
19.9.4	Monitoring the Feedback and Result . . . . .	205
19.9.5	Shutting Down the Nodes . . . . .	206
19.9.6	Things to Keep in Mind . . . . .	206
19.9.7	Summary of New Commands and Functions . . .	207
19.10	Conclusion . . . . .	207
<b>20</b>	<b>ROS Functions &amp; Commands</b>	<b>209</b>
	<b>Reference</b>	<b>217</b>
	<b>Bibliography</b>	<b>227</b>

# List of Figures

1.1	Analogy of Reinventing the Wheel vs. Building on Existing Work . . . . .	21
1.2	Robotic Development Workflow: Even a simple task like a blinking light requires numerous steps. . . . .	23
4.1	Nodes communicate through the ROS Master . . . . .	41
4.2	Nodes Communicating with Each Other through the ROS Master . . . . .	42
4.3	Node Lifecycle in ROS . . . . .	43
4.4	Publisher-Subscriber Communication in ROS . . . . .	44
4.5	Multiple Nodes Communicating via Topics in ROS . . . . .	45
4.6	Analogy: Departments (Nodes) Communicate Through Mailing System (Topics) . . . . .	46
4.7	Client-Server Communication for Services . . . . .	47
4.8	Action Client-Server Communication . . . . .	48
14.1	Project Workflow Diagram . . . . .	121
18.1	Client-Server Communication Diagram . . . . .	159
19.1	Action Communication Workflow . . . . .	179



# What is ROS?



## 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.

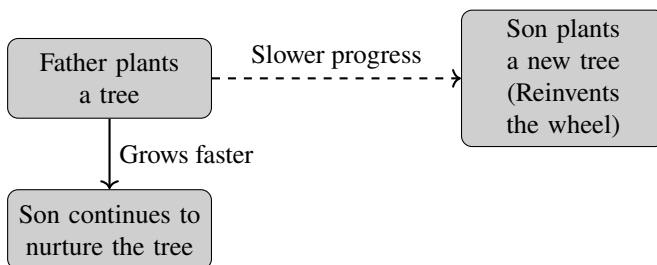


Figure 1.1: Analogy of Reinventing the Wheel vs. Building on Existing Work

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.

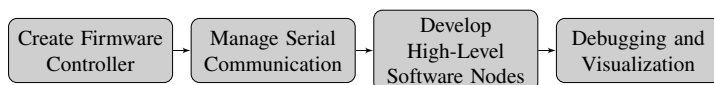


Figure 1.2: Robotic Development Workflow: Even a simple task like a blinking light requires numerous steps.

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!

