C++ for Robotics Engineering (Foundations to ROS2 Implementation)

Course Description

This course provides a structured pathway for experienced software professionals transitioning into robotics software engineering using **C++ and ROS2**. The training focuses on developing strong modern C++ programming skills required in robotics systems and applying them in real robotics software architectures using ROS2.

Participants begin by refreshing core C++ concepts with a robotics-oriented perspective and gradually move into advanced topics such as memory management, RAII, smart pointers, move semantics, concurrency, and debugging techniques. The course emphasizes writing safe, efficient, and production-quality modern C++ code.

The program then transitions into robotics application development using **ROS2 with C++ (rclcpp)**. Learners will develop ROS2 nodes, implement publishers, subscribers, services, and actions, and understand system-level architecture patterns such as executors, callback groups, plugin-based architectures, navigation stack integration, and behavior tree systems.

Hands-on labs and implementation exercises form the core of the training. The program concludes with a **capstone robotics software project**, where participants develop a complete ROS2-based robotics component demonstrating real-world robotics engineering capability.

Audience Profile

This course is designed for:

- Software engineers transitioning into robotics engineering roles
- Robotics and AI graduates who want to strengthen C++ programming skills
- Developers with Python robotics experience who need C++ capability for robotics jobs
- Engineers interested in ROS2-based robotics software development
- Professionals aiming to work on robotics middleware, autonomy systems, or robot software stacks

---

Prerequisite

Essential

- Programming experience in any object-oriented language
- Basic command-line familiarity
- Basic understanding of Linux environments

Recommended

- Prior exposure to robotics concepts
- Basic understanding of Python programming
- Familiarity with Ubuntu or Linux-based systems

Course Objective

By the end of this course, participants will be able to:

- Write, debug, and structure production-quality **modern C++ robotics code**
- Understand and apply modern C++ concepts including **RAII, smart pointers, move semantics, STL, and concurrency**
- Build and manage **C++ projects using CMake**
- Debug complex C++ applications using tools such as **gdb and sanitizers**
- Develop robotics applications using **ROS2 C++ (rclcpp)**
- Implement **publishers, subscribers, services, and actions** in ROS2
- Design scalable robotics software using **executors, callback groups, and plugin architectures**
- Integrate custom components into the **ROS2 Navigation stack (Nav2)**
- Extend robotics behavior using **behavior trees**
- Implement **sensor processing pipelines** in ROS2 systems
- Build and demonstrate a **complete robotics software component using ROS2 and C++**

Table of Contents (TOC)

Phase 0 — Environment Setup and Baseline Readiness

Module 1: Development Environment Setup

- Ubuntu Linux environment preparation
- Installing compiler toolchains (g++ / clang)
- Build system setup using CMake
- Development environment configuration using VS Code
- Introduction to project repository and workspace structure

Module 2: Debugging and Diagnostic Tools Setup

- Installing and configuring gdb
- AddressSanitizer setup and usage
- ThreadSanitizer setup and usage
- Running and compiling a sample C++ project

Phase 1 — C++ Fundamentals (Robotics-Oriented)

Module 3: C++ Program Model and Compilation

- Structure of a C++ program
- Header files vs source files
- Translation units and linking
- Namespaces
- Input and output operations
- Control flow constructs

Module 4: Types, References, Pointers and Const Correctness

- Primitive types and type conversions
- References and pointers
- Const correctness principles
- Top-level vs low-level const

- Type deduction using auto and decltype

Module 5: Standard Template Library Foundations

- std::string usage and manipulation
- std::vector container usage
- Iterators and iteration patterns
- Associative containers: std::map and std::unordered_map
- Algorithm library usage (std::algorithm)
- erase-remove idiom
- Complexity awareness and Big-O considerations

Phase 2 — Advanced C++ Core (Ownership and Memory Safety)

Module 6: Object Lifetime and RAII

- Stack and heap allocation
- Deterministic destruction in C++
- Resource Acquisition Is Initialization (RAII)
- Smart pointers overview
- std::unique_ptr usage
- std::shared_ptr and reference counting
- std::weak_ptr usage and ownership cycles
- Ownership transfer patterns

Module 7: Constructors, Destructors and Move Semantics

- Constructor initialization lists
- Copy constructors and copy assignment
- Move constructor and move assignment
- Rule of Five design principle
- Avoiding unnecessary copying

- Performance implications of move semantics

Module 8: Exception Handling and Safety

- Exception handling mechanisms in C++
- try, catch, and exception propagation
- noexcept specification
- Exception safety guarantees

Phase 3 — Modern C++ for Robotics Codebases

Module 9: Modern C++ Language Features

- Type deduction with auto
- Range-based loops
- Lambda expressions and functional patterns
- Strongly typed enumerations (enum class)
- override and final specifiers
- =delete and =default usage
- Structured bindings
- std::optional and std::variant
- constexpr for compile-time computation

Phase 4 — Concurrency, Performance and Debugging

Module 10: Concurrency Foundations

- Multithreading concepts
- std::thread usage
- Mutex mechanisms and synchronization
- std::lock_guard and std::unique_lock
- Atomic operations with std::atomic

- Race conditions and thread safety
- Deadlock detection and prevention

Module 11: Performance Optimization in C++

- Copy vs move performance implications
- Pass-by-value vs pass-by-reference
- Efficient container usage with reserve()
- Avoiding dynamic allocations in loops
- Cache locality fundamentals

Module 12: Debugging and Diagnostics

- Debugging C++ applications with gdb
- Understanding stack traces
- Memory debugging using AddressSanitizer
- Concurrency debugging using ThreadSanitizer
- Identifying and fixing memory leaks

Phase 5 — C++ for Robotics (ROS2 Implementation)

Module 13: ROS2 Architecture and C++ Node Development

- ROS2 architecture overview
- ROS2 workspace and package structure
- Writing ROS2 nodes using rclcpp
- Publishers and subscribers
- Timers and callback mechanisms
- Service interfaces
- Parameter handling
- Logging and shutdown handling

Module 14: Executors and Callback Architecture

- ROS2 executor model
- SingleThreadedExecutor usage
- MultiThreadedExecutor usage
- Callback groups
- Designing non-blocking node architectures

Module 15: ROS2 Communication Patterns

- Message design principles
- Quality of Service (QoS) configuration
- Services vs actions usage patterns
- Parameter management strategies

Module 16: Plugin-Based Architecture using pluginlib

- Plugin architecture in robotics systems
- Plugin interfaces and abstraction
- Runtime plugin loading
- Planner and controller plugin patterns

Module 17: Navigation Stack Integration (Nav2)

- Nav2 system architecture
- Planner interfaces
- Controller interfaces
- Parameter configuration and tuning
- Integration of custom components

Module 18: Behavior Tree Integration

- Behavior tree architecture
- Behavior tree nodes and execution flow
- Integration with navigation stack
- Extending behavior trees with custom nodes

Module 19: Sensor Processing Node Patterns

- Sensor data pipelines
- Message buffering techniques
- Data filtering approaches
- Thread-safe data processing

Capstone Project — Robotics Software System Implementation

Module 20: Robotics Software Capstone Project

- Designing a robotics software component architecture
- Implementing ROS2 nodes using C++
- Publisher and subscriber integration
- Services and action interfaces
- Multi-threaded executor usage
- Plugin-based component design
- Navigation stack integration
- Behavior tree customization
- Debugging using gdb and sanitizers
