

1. For the text below, summarise the problem and suggest a software system architecture solution. The main stakeholders in this case are: - Software system architects - DevOps engineers - Developers

Problem, context, and related work

Robot Operating System 2 (ROS) is a popular and widely adopted robotics engineering framework. Its main goal is to provide robotics engineers a platform that allows for the development of robotics applications. Framework's infrastructure, which is open source, is based on numerous distributed packages. ROS supports a wide range of devices, starting from small embedded devices up to autonomous industry robots. ROS was developed with real-time capabilities in mind. Since this technology is in its infancy, the ROS community encourages industry and academic research. Academic research in ROS has covered many aspects of robotics development and has achieved quantifiable results regarding soft and hard real-time constraints, lowering latency, and in general, improving performance. However, because of its complicated vertical stack of technologies, there is still room where significant performance benefits could be extracted. The default way to extract performance with aim to lower execution time is by utilizing multi-cores. However, ROS with its complex infrastructure and requirement from robotics engineers to learn parallelization from C/C++ perspective, make it hard to utilize multi-cores properly. Therefore, utilizing multi-cores to lower execution time is one of the main motivators that guides research in ROS. By lowering execution time we gain performance benefits and allow other parts of ROS infrastructure, which are influenced by latency and real-time constraints, to be positively impacted as well. Robots are very heterogeneous devices. Therefore, robots may come with a vast array of different system requirements and system constraints. Some of the system constraints are critical when it comes to the correct functionality of the robots. These critical constraints deal with the overall performance of the robotics application. In the scope of overall performance of the robotics application, the main goal of robotics engineers is to lower execution time or make a particular function execute faster. To facilitate this, we refer to utilization of multi-core processors. However, required specialized knowledge regarding concurrency and expertise, which robotics engineers do not have, slows them down in building their robotics application and decreases efficiency. Robotics engineers are required to understand the architecture of the CPU and how it interacts with main memory. On top of that robotics engineers are required to understand the intricacies regarding caches and data locality. Complemented with this is a software stack that is utilizing the CPU. They need to learn how to use multi-threading libraries, which is not simple. Functions for creating and managing threads require thinking about multiple data-flows in software stack. As a consequence, robotics engineers are required to understand in low-level detail how multi-threading works and how prioritization, scheduling, and affinity combine to solve the problem of lowering execution time. To conclude, the main gap is that robotics engineers do not have an intuitive way of utilizing multi-cores to be able to lower execution time. They are required to know low-level details regarding parallelization and have to develop multi-threading applications. Additionally, robotics engineers do not have accessible interfaces to control Operating System mechanisms regarding prioritization, scheduler type, and affinity

Requirements

- SR.01: Framework to facilitate development of multithreaded applications in ROS2
- SR.02: Framework to assist robotics engineers in developing software that takes advantage of multi-cores in order to reduce average execution time, integrated in ROS2.
- SR.02: Pre-locking heap memory - The framework enables pre-locking of heap memory during initialization of the application in order to avoid failed memory allocations, which can occur if application requires significant amount of memory.
- SR.03: Process priority configuration - The framework provides interface for setting priorities of threads in order to avoid priority inversion.
- SR.04: Measurement of heap allocations - The framework offers a visualization on how many memory allocations happened, and it should be easily accessible to the robotics engineers
- SR.05: Measurement of latency. - The framework has a mechanism that

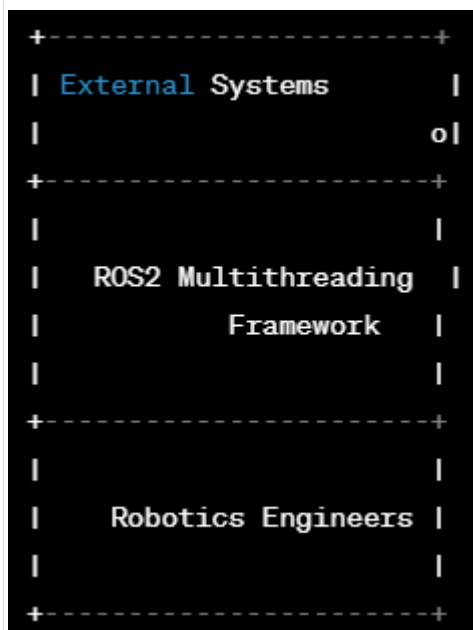
measures latency • SR.06: Measurement of jitter - The framework has a mechanism that measures jitter • SR.07: Measurement of CPU caches usage & The framework has a mechanism that measures CPU cache usage • SR.08: Measurement of memory fetches. - The framework has a mechanism which measures memory fetches. • SR.09: Executor type - Robotics engineers write nodes' configuration as input to the framework. The framework outputs the best performing executor type configuration containing nodes. & ROS executor • SR.10: Node to process assignment - Robotics engineers write nodes' configuration as input to the framework. The framework outputs the best performing configuration. & node assignment, thread, process • SR.11: Allocator type - The framework accepts the message data types that are used throughout the application and it advises robotics engineers on which allocators to use. • SR.12: Easy to use user interface - All information and capabilities of the framework should be gathered in easy-to-use user interface. • SR.13: Guideline - Robotics engineers will be able to express their intentions of parallelizing the code in the framework. The framework will output guidelines for robotics engineers. • SR.14: Parallel Configuration - The framework will accept non parallelized code and will turn it into parallel configuration. & Parallel Configuration Quality requirements • SR.01: Increased CPU Utilization - The framework facilitates the development of multi-threaded applications, and as a result, increased CPU utilization. If the number of cores are n , (and $n > 1$), then utilization is at least $0.7 * n$. • SR.02: Decreased Heap Allocation - The framework facilitates the mechanism with pre-locking memory during the start of the execution of the application. At least 50 percent decrease in heap allocations regarding memory is expected. • SR.03: Increased general performance of the application - The framework facilitates the mechanism for process priority configuration. We expect an increase in general performance in a region of 10 percent. • SR.04: Decreased average execution time - The framework facilitates the mechanism for decreasing average execution time by utilizing multi-threading. • SR.05: Increased CPU Cache utilization - We expect an increase in CPU Cache utilization by 30 percent. • SR.06: Decreased Memory Fetches - We expect to have a decreased memory fetches by 20 percent as a result of increased CPU Cache utilization. • SR.07: Increased productivity of robotics engineers - The facilities of the framework will output parallelized configurations. Compared with by-hand optimization, this process will be expected to take 30 percent less time. & increased productivity • SR.08: Usability - The framework will include an easy to use user interface. We expect robotics engineers to quickly understand the framework and use it efficiently. • SR.09: Modularity - The framework's design will closely follow ROS design philosophy. We expect the framework to be at least modular as ROS and to fit on top of ROS. & modularity Use cases • UC.01: Direct Inverse Kinematics - Sensors, Actuators, Moveable joints, Rotating Joints & Inverse kinematics is a field which concerns itself with correct positioning of the joints in order for end-actuator to reach desired position. As the name indicates, the positioning is worked out from the end-actuator backwards to the first-actuator. • UC.02: Detecting human faces and torso orientation - People in a scene, robot scanning the scene - Image processing is a very important field in robotics engineering. This use case tries to identify the way of doing image processing in robotics, so we can learn how to parallelize common functions. • UC.03: Kalman Filter & A specific robot does sensor fusion and uses kalman filter for prediction - Prediction of the measurements. Kalman filtering is a mathematical approach on how to improve our measures that come from the sensors. • UC.04: Assessing execution time of real-time loop - Real-Time Loops, Function, Time & Since ROS was built with real-time capabilities in mind, measuring real-time loop execution time can give us an insight on performance improvements.

The problem is that robotics engineers using Robot Operating System 2 (ROS) have difficulty utilizing multi-core processors to reduce execution time due to the complex infrastructure of ROS and the need for low-level knowledge of parallelization. The solution is to develop a framework integrated into ROS2 that assists robotics engineers in developing multithreaded applications to

take advantage of multi-cores. The framework should include features such as pre-locking heap memory, process priority configuration, measurement of heap allocations, latency, jitter, CPU cache usage, and memory fetches. It should also provide a user-friendly interface and output guidelines for parallelizing code. The quality requirements include increased CPU utilization, decreased heap allocation, increased general performance, decreased average execution time, increased CPU cache utilization, decreased memory fetches, increased productivity of robotics engineers, usability, and modularity. The use cases include direct inverse kinematics, detecting human faces and torso orientation, Kalman filter, and assessing execution time of real-time loop.

2. Suggest architecture of a conceptual solution for this problem. We need a C4 system context diagram (<https://c4model.com/#SystemContextDiagram>). Please use text (arrows, dots, different shapes, and different colours) to visualise the solution. Consider the following: - Identify relevant external systems (if any) and add them to the diagram. - Identify relevant external stakeholders (if any) and add them to the diagram. - Identify high level abstract data types transferred between components and stakeholders. - Summarise data types transmitted between components. Add this information to the connections and interfaces between components.

Here is a possible C4 system context diagram for the conceptual solution:



The diagram shows the ROS2 Multithreading Framework as the main component, with Robotics Engineers as the only external stakeholder. External Systems could be added as necessary, depending on the specific use case.

The data types transmitted between the components would include:

- Configuration data: This would be data describing the nodes, threads, processes, and allocators used in the robotics application. This data would be input by Robotics Engineers and processed by the Framework to output optimal configurations for execution.
- Performance data: This would be data describing the latency, jitter, CPU cache usage, memory fetches, and heap allocations of the robotics application during execution. This

data would be output by the Framework and displayed to Robotics Engineers for analysis and optimization.

- ROS message data: This would be data transmitted between nodes in the ROS infrastructure. The Framework would need to be compatible with these messages and advise Robotics Engineers on which allocators to use for efficient memory management.

The connections and interfaces between components would be:

- Robotics Engineers -> ROS2 Multithreading Framework: Configuration data input.
- ROS2 Multithreading Framework -> Robotics Engineers: Parallelization guidelines output, performance data visualization output.
- ROS2 Multithreading Framework -> ROS2: Configuration data input, ROS message data output.
- ROS2 -> ROS2 Multithreading Framework: ROS message data input.

3. Suggest architectural components that we need to solve this problem. We need a C4 container solution diagram (<https://c4model.com/#ContainerDiagram>). Please use text (arrows, dots, different shapes, and different colours) to visualise the solution. Consider the following: - Define how the main stakeholder will use the solution. Add it to the diagram.. - Which quality properties are important? Suggest adequate architectural solutions to address them (architectural style, design patterns, design tactics, components and their relationships). Structure the components according to it. - Define data types transferred between components. - Summarise data types transmitted between components. Add this information to the connections and interfaces between components. - Define interfaces between components. Add information about required and provided interfaces to the components, along with the data types. - Document which anti-patterns we need to avoid when implementing this solution.

Here is a possible C4 container diagram for the solution:



The main stakeholder, Robotics Engineers, will use the solution by providing their robotics application code to the ROS2 Multithreading Framework as input, along with configuration data. The Framework will then process this input to output optimized configurations for execution, along with parallelization guidelines and performance data visualization.

Important quality properties for the solution include increased CPU utilization, decreased heap allocation, increased general performance, decreased average execution time, increased CPU cache utilization, decreased memory fetches, increased productivity of robotics engineers, usability, and modularity. The architectural solutions to address these properties include:

- Multithreading design pattern: The Framework should use this pattern to take advantage of multi-core processors and reduce average execution time.
- Pre-locking heap memory: The Framework should provide a mechanism to pre-lock heap memory during initialization of the application in order to avoid failed memory allocations.
- Process priority configuration: The Framework should provide an interface for setting priorities of threads in order to avoid priority inversion.
- Measurement of performance data: The Framework should provide a mechanism for measuring latency, jitter, CPU cache usage, and memory fetches to aid in optimization.
- User-friendly interface: The Framework should include an easy-to-use user interface to facilitate usability.
- Modularity: The Framework should be designed to fit on top of the ROS infrastructure and be at least as modular as ROS.

Data types transferred between components include:

- Configuration data: This would be data describing the nodes, threads, processes, and allocators used in the robotics application.
- Performance data: This would be data describing the latency, jitter, CPU cache usage, memory fetches, and heap allocations of the robotics application during execution.
- ROS message data: This would be data transmitted between nodes in the ROS infrastructure.

The connections and interfaces between components would be:

- Robotics Engineering Tool -> ROS2 Multithreading Framework: Configuration data input.
- ROS2 Multithreading Framework -> Robotics Engineering Tool: Parallelization guidelines output, performance data visualization output.
- ROS2 Middleware -> ROS2 Multithreading Framework: ROS message data input/output.
- Operating System -> ROS2 Middleware: Low-level system interaction.

The interfaces between components would include:

- ROS2 Multithreading Framework: Requires configuration data input and provides parallelization guidelines output and performance data visualization output.
- ROS2 Middleware: Requires ROS message data input/output.
- Robotics Engineering Tool: Requires parallelization guidelines output and performance data visualization output.

Anti-patterns to avoid when implementing this solution include:

- Big ball of mud: This should be avoided by ensuring that the Framework is modular and well-organized.
- Golden hammer: This should be avoided by selecting the appropriate design patterns and components for the solution based on the specific requirements and use cases.

4. Suggest architectural components that we need to solve this problem. We need a C4 component solution diagram (<https://c4model.com/#ComponentDiagram>). Please use text

(arrows, dots, different shapes, and different colours) to visualise the solution. Consider following:

- Define how the main stakeholder will use the solution. Add it to the diagram.
- Which quality properties are important? Suggest adequate architectural solutions to address them (architectural style, design patterns, design tactics, components and their relationships). Structure the components according to it.
- Define data types transferred between components.
- Summarise data types transmitted between components. Add this information to the connections and interfaces between components.
- Define interfaces between components. Add information about required and provided interfaces to the components, along with the data types.
- Document which anti-patterns we need to avoid when implementing this solution.

Here is a possible C4 component diagram for the solution:

```
+-----+
|  Robotics Engineering Tool  |
|                               |
+-----+-----+
|
|  uses
|
|
+-----v-----+
| ROS2 Multithreading Framework|
|                               |
+-----+-----+
|
|  uses
|
|
+-----v-----+
|  Executor Configuration      |
|      Component              |
+-----+-----+
|
|  provides
|
|
+-----v-----+
|  Node Configuration         |
|      Component              |
+-----+-----+
|
|  provides
|
|
+-----v-----+
|  Allocator Configuration    |
|      Component              |
+-----+-----+
|
|  provides
|
|
+-----v-----+
|  Performance Data Visualizer|
|      Component              |
+-----+-----+
```


The main stakeholder, Robotics Engineers, will use the solution by providing their robotics application code to the ROS2 Multithreading Framework as input, along with configuration data for the executor, nodes, and allocators to use in the application. The Framework will then process this input to output optimized configurations for execution, along with parallelization guidelines and performance data visualization.

Important quality properties for the solution include increased CPU utilization, decreased heap allocation, increased general performance, decreased average execution time, increased CPU cache utilization, decreased memory fetches, increased productivity of robotics engineers, usability, and modularity. The architectural solutions to address these properties include:

- Multithreading design pattern: The Framework should use this pattern to take advantage of multi-core processors and reduce average execution time.
- Pre-locking heap memory: The Framework should provide a mechanism to pre-lock heap memory during initialization of the application in order to avoid failed memory allocations.
- Process priority configuration: The Framework should provide an interface for setting priorities of threads in order to avoid priority inversion.
- Measurement of performance data: The Framework should provide a mechanism for measuring latency, jitter, CPU cache usage, and memory fetches to aid in optimization.
- User-friendly interface: The Framework should include an easy-to-use user interface to facilitate usability.
- Modularity: The Framework should be designed to fit on top of the ROS infrastructure and be at least as modular as ROS.

Data types transferred between components include:

- Configuration data: This would be data describing the executor, nodes, and allocators used in the robotics application.
- Performance data: This would be data describing the latency, jitter, CPU cache usage, memory fetches, and heap allocations of the robotics application during execution.

The connections and interfaces between components would be:

- Robotics Engineering Tool -> ROS2 Multithreading Framework: Configuration data input.
- ROS2 Multithreading Framework -> Executor Configuration Component: Executor configuration data output.
- ROS2 Multithreading Framework -> Node Configuration Component: Node configuration data output.
- ROS2 Multithreading Framework -> Allocator Configuration Component: Allocator configuration data output.
- ROS2 Multithreading Framework -> Performance Data Visualizer Component: Performance data visualization output.

The interfaces between components would include:

- ROS2 Multithreading Framework: Requires configuration data input and provides executor, node, and allocator configuration data output and performance data visualization output.

- Executor Configuration Component: Provides executor configuration data output.
- Node Configuration Component: Provides node configuration data output.
- Allocator Configuration Component: Provides allocator configuration data output.
- Performance Data Visualizer Component: Requires performance data input.

Anti-patterns to avoid when implementing this solution include:

- God object: This should be avoided by breaking down the functionality into smaller, more focused components.
- Spaghetti code: This should be avoided by ensuring that the components are well-organized and

5. Please suggest a toolchain to implement the solution. Consider which tools we can reuse. For the components we need to develop, please suggest a programming language. Please use text (arrows and dots) to visualise the solution. Define data types transferred between components. Summarise data types transmitted between components. Add this information to the connections and interfaces between components.

A possible toolchain for implementing the solution could include the following:

- Programming language: C++
- ROS2: This can be used as the underlying robotics engineering framework.
- OpenCV: This can be used for image processing in UC.02.
- Performance profiling tools: These can be used to measure and analyze the performance data generated by the solution.

Here is a possible visualization of the toolchain:

```
+-----+
|      Robotics      |
|      Engineering   |
|      Tool          |
+-----+
|
|  uses
|
|
+-----v-----+
|  ROS2 Multithreading |
|      Framework      |
|  (C++ Programming)  |
+-----+
|
|  uses
|
|
+-----v-----+
|  Executor Configuration |
|      Component         |
|  (C++ Programming)     |
+-----+
|
|  provides
|
|
+-----v-----+
|  Node Configuration    |
|      Component         |
|  (C++ Programming)     |
+-----+
|
|  provides
|
|
+-----v-----+
|  Allocator Configuration |
|      Component         |
|  (C++ Programming)     |
+-----+
|
|  provides
|
|
+-----v-----+
|  Performance Data Visualizer |
|      Component         |
|  (C++ Programming)     |
+-----+
```

Data types transferred between components would include:

- Configuration data: This would be data describing the executor, nodes, and allocators used in the robotics application. This data would likely include data types such as integers, booleans, and strings.
- Performance data: This would be data describing the latency, jitter, CPU cache usage, memory fetches, and heap allocations of the robotics application during execution. This data would likely include data types such as floating point numbers, integers, and booleans.

The connections and interfaces between components would be:

- Robotics Engineering Tool -> ROS2 Multithreading Framework: Configuration data input.
- ROS2 Multithreading Framework -> Executor Configuration Component: Executor configuration data output.
- ROS2 Multithreading Framework -> Node Configuration Component: Node configuration data output.
- ROS2 Multithreading Framework -> Allocator Configuration Component: Allocator configuration data output.
- ROS2 Multithreading Framework -> Performance Data Visualizer Component: Performance data visualization output.

The interfaces between components would include:

- ROS2 Multithreading Framework: Requires configuration data input and provides executor, node, and allocator configuration data output and performance data visualization output.
- Executor Configuration Component: Provides executor configuration data output.
- Node Configuration Component: Provides node configuration data output.
- Allocator Configuration Component: Provides allocator configuration data output.
- Performance Data Visualizer Component: Requires performance data input.

Overall, the toolchain would use C++ programming for the components, with ROS2 as the underlying robotics engineering framework and OpenCV for image processing in UC.02. Performance profiling tools would also be used to measure and analyze performance data generated by the solution.