Improve the Autoware Launch System #5313

xmfcx started this conversation in **Design**

xmfcx on Oct 8 Maintainer edited

Category
Design

Labels
DevOps Dojo: ROS ...
component:launch

5 participants

Improve the Autoware Launch System

Current launch system and its issues

Depth and dependencies

Launch system is composed of too many repositories and complex dependencies.

Example pathway:

Let's analyze the launch file used for running the Autoware with rosbag replay sim demo for the lidar pipeline.

#	Launch File	Package Name
1	logging_simulator.launch.xml#L39	autoware_launch
2	autoware.launch.xml#L89	autoware_launch
3	tier4_sensing_component.launch.xml#L3	autoware_launch
4	sensing.launch.xml#L13	tier4_sensing_launch
5	sensing.launch.xml#L9	sample_sensor_kit_laur
6	lidar.launch.xml#L14	sample_sensor_kit_laur
7	velodyne_VLS128.launch.xml#L18	common_sensor_launch
8	nebula_node_container.launch.py#L165	common_sensor_launch

This is 8 Levels deep. Across 3 separate repositories, 4 packages.

Loose coupling and dependencies

We have some repository names depending on these 2 global variables:
https://github.com/autowarefoundation/autoware_launch/blob/80a387e10c6a2615d3cb45854fecc5137c5a68f4/autoware_launch/launch/autoware.launch.xml#L5-L6

```
<arg name="vehicle_model" default="sample_vehicle"/>
```

```
<arg name="sensor_model" default="sample_sensor_kit"/>
```

These names are loosely used in variables and expected to be part of the package names.

- https://github.com/autowarefoundation/sample_sensor_kit_launch
- https://github.com/autowarefoundation/sample_vehicle_launch

And these vague names are referred in many places:

▶ Click here to expand

These complicate things too much. Makes it very hard to navigate or understand how things are launched.

Configuration file paths

We have duplication of `package_name.param.yaml` in both autoware_launch and autoware.universe and many other places.

https://github.com/search?q=org%3Aautowarefoundation+.param.yaml+path%3A*.launch.xml&type=code

Sometimes they are from the packages themselves, sometimes they are from the launch system.

Parameter/argument scope bleeding

ROS 2 launch system allows passing all the arguments downstream by default.

Simple example:

- https://github.com/autowarefoundation/autoware_launch/blob/cf9840b14d136d48e3e2ba8529163cff16ab1fa5/autoware_launch/launch/autoware.launch.xml#L89
  - Here a launch file is called without passing it any arguments explicitly.
- https://github.com/autowarefoundation/autoware_launch/blob/cf9840b14d136d48e3e2ba8529163cff16ab1fa5/autoware_launch/launch/components/tier4_sensing_component.launch.xml#L3
  - But here we can see it can actually make use of the arguments from the higher scope without explicit passing.

This is a very dangerous practice and leads to very hard to debug issues.

There is probably a way to limit this scope but I don't know how to do it. There is `<group scoped="true"> </group>` but it solves a different problem. (since `humble`, this is `true` by default anyways.)

General problems

- We don't know what composable nodes, nodes, component containers we are attempting to launch.

    - To understand this, we need to go through the launch files manually and see what is being launched.
    - Good luck doing this with all the depth and breadth of the launch system.
- We don't know what parameters are being passed to these nodes in the end.
    - The parameters are scattered all over the place.
    - Sometimes they are overridden downstream.
    - Sometimes some are provided by the param.yaml files, some are provided by the launch files directly.
- Because we have inter-dependent repositories, when a node is updated, we need to update the launch and param files in multiple repositories.
    - Sometimes deveolpers overlook some of these and we end up with broken launch files.

Summary of current issues

1. **Depth and Dependencies:**

    - The launch system has too many nested files across multiple repositories.
    - Example scenario shows 8 levels deep across 4 packages and 3 repositories.
    - Makes understanding the launch sequence challenging.

2. **Loose Coupling and Dependencies:**

    - Use of global variables for vehicle and sensor models leads to vague dependencies.
    - These variables are referenced in multiple places across packages, increasing complexity.

3. **Configuration File Duplication:**

    - Multiple `.param.yaml` files exist in different locations (e.g., `autoware_launch`, `autoware.universe`).
    - This duplication causes confusion about which file is being used and where.

4. **Parameter/Argument Scope Bleeding:**

    - Arguments are passed downstream by default in ROS 2, making debugging difficult.
    - Launch files can use arguments from higher scopes without explicit declaration, leading to scope issues.

5. **General Problems:**

    - Difficulty identifying which nodes, composable nodes, and component containers are being launched.
    - Parameters are scattered across launch files, making them hard to track and debug.

- Inter-repository dependencies mean updates require changes across multiple locations, leading to potential oversights.

Expectations for the new launch system

1. Centralize all launch files into a single, separate repository.
2. Simplify the launch file hierarchy to reduce depth and complexity.
3. Use explicit argument passing to prevent parameter scope bleeding.
4. Eliminate duplication of configuration files across repositories.
5. Consolidate all configuration files into a centralized location within the launch repository.
6. Decouple inter-repository dependencies by ensuring launch files are self-contained.
7. Implement (CI) to automatically test and validate launch files.

Launch Unifier to analyze and simplify

▶ Click here to expand 🔛

Slides

Launch Unifier Tool

M. Fatih Cırıt - Software Architect

mfc@leodrive.ai

https://github.com/xmfcx/launch_unifier_ws

https://github.com/xmfcx/one_launch

I suggest using and improving this tool to simplify the current launch setup, making it easier to manage and more efficient.

I've shared my perspective, I'm happy to hear your ideas and potential solutions.

cc. @mitsudome-r @yukkysaito @TakaHoribe @esteve @kaspermeck-arm @samet-kutuk @oguzkaganozt @HamburgDave @youtalk

↑ 6 🚀 2 👀 2

3 comments · 3 replies

Oldest | Newest | Top

amadeuszsz on Oct 16 Collaborator

I'm big fan of this idea 🚀 . The current launch system takes too long to start up.

I just have some preliminary thoughts, which I would like to share with you.

Mostly agree. I think we can remove all launch files and configuration files from `feature_x`-like and `tier4_<component>_launch` packages and move them to the top launch package.

However, the hierarchy of launch files in such a large system is appropriate approach, because:

- Allows for reuse high abstracted launch files in various scenarios.
- More readable if we can divide component for smaller parts and load them based on current configuration.
- Maintainers can just run `feature_x`-like launch files and quickly test feature without necessity of launching whole stack. Here I can imagine launch & configuration files in top launch repository, even though they are not used in any of launch scenario.
- If `launch_unifer` can quickly generate unified single launch file, do we need to downgrade the hierarchy?

Moving all launch & configuration files to one repository will help with future launch testing and make Autoware configuration clearer, especially for newcomers.

I'm aware that we might be far away from final solution yet, but at the end I see integration of launch unifer on two ways:

- Wrapper for current `ros2 launch` CLI, e.g. `autoware launch`, where we generate unified launch file and silently launch it via standard `ros2 launch` CLI. The file generation can follow ROS logging, where we store files somewhere in `~/.ros` directory.
- Same as first option, but simply handle it via script execution in top launch file - `ros2 launch` CLI applies here.

↑ 1 2 replies

xmfcx on Oct 18 Maintainer Author

> However, the hierarchy of launch files in such a large system is appropriate approach

@amadeuszsz I agree with this opinion. I do not propose to reduce all into a single launch file.
It would be a nightmare to maintain all in a single file.

But I also don't want it to be 8-9 levels deep. I think limiting it to 4-5 levels is enough.

https://github.com/xmfcx/one_launch/tree/feat/add-sensing/one_launch/launch

If you look at my launch file's sensing part, it goes like:

`one_launch->sensing->lidar->vlp16->velodyne_node_container`

and that's it.

And maybe could be simplified more.

`launch_unifier` is there to show the limits and help debugging etc.

mitsudome-r on Oct 21 Maintainer

Just FYI, it seems like most of the startup time latency comes from this part of the code in ros_launch:
TakaHoribe/launch@ `a1fcab0`

It is a section that validates if all arguments are defined properly at runtime. Maybe we could also look deeper look into this function and see if there are any optimization we can do.

According to **@TakaHoribe**, the followings were the results of startup time:

- default planning_simulator.launch.xml: about 19 seconds
- commenting out the validation: about 6 seconds
- using onelaunch: about 5 seconds

technolojin on Oct 20 Collaborator

I agree that the autoware launch system need to be changed.

In my opinion, writing ROS launcher of this complex system "by hand" is the problem.
It is a small miracle that we can launch the system with this hand-crafted launcher.

We have to clarify rule of what we call "launcher" today.

1. Software architecture: structure of software modules that consist the system
2. Parameters: conditions to run the software (including ml models)
3. Interfaces: map of the ROS topics(external interface) to the software module in/outs (internal interface). <the concept is implemented here>
4. Launcher: script that launch the software modules

To solve this problem, I would like to propose a new approach.

1. We(humans) describe structure(SW architecture) of the autoware of desired systems (an architecture per deployment)
2. We also pairs a parameter set to an architecture

3. Computer decodes (or let say compile) the architecture (The topology of the system will be checked, and we will know if the system can be launched in this step!)
4. Interfaces (the ROS topic mapping) is determined
5. The launchers are generated

I know this is a big change. But, I do not think we can solve the complex and inefficient situation of the current autoware launcher.

↑ 1 0 replies

technolojin on Oct 20 Collaborator

1. Centralize all launch files into a single, separate repository.
   a. the launcher will be generated
2. Simplify the launch file hierarchy to reduce depth and complexity.
   a. we will describe architecture per node, sub-module (set of nodes, ex. lidar pipeline), module (set of sub-modules, ex. perception) in hierarchy structure
3. Use explicit argument passing to prevent parameter scope bleeding.
   a. Parameter sets are described for nodes, and the architecture will treats the parameters as a type of input
   b. The parameter sets will be connected as input to the SW elements
   c. The topology and list will be checked in the compiling process
4. Eliminate duplication of configuration files across repositories.
   a. May the definition of the parameters will be with the nodes, but the user may want to overwrite those values
   b. Only the overwriting parameters will be set and contained to a specific deployment
   c. (Because there are needs to have different parameter for users system, configuration will be in multiple repositories. But not all of them like now)
5. Consolidate all configuration files into a centralized location within the launch repository.
   a. The final parameters will be gathered when the architecture is compiled
   b. May the final parameters are not contained in a repository, but in the products (ex. build/install folder)
6. Decouple inter-repository dependencies by ensuring launch files are self-contained.
   a. The nodes, sub-modules, modules will be distributed over the repositories, and will be self-contained
   b. Each step of the architecture hierarchy will act as an abstract layer, so that the higher layer do not need to know changes of lower layers
   c. The architecture of the system is dependent to specific combination of the repositories, but minimum alignment is required
7. Implement (CI) to automatically test and validate launch files.
   a. Topology of the system will be checked by the architecture compiler
   b. The compiler will be triggered in local build, also in CI

c. Developers need to describe their developing software component in the packages in a right way to pass the compiling process

↑ 1 1 reply

mojomex 3 weeks ago Collaborator edited

For parameters, I think we should annotate e.g. our JSON schemas with the type of parameter:

- vehicle_model dependent
- sensor_kit dependent
- intrinsic to the node, e.g. a tuneable parameter for a detection model
- parameter exposed as a launch parameter, e.g. map path, wether to connect to real hardware, etc.
- etc.

Also, sometimes a set of parameters will change based on another parameter, e.g.:

- depending on PTP protocol version, the parameters of PTP change
- see this JSON schema for how to model that nicely.

Respecting this in the launch generator could eliminate parameter config errors during launch file generation, eliminating annoying launch-time / runtime errors.

For reference: Autoware docs for JSON schema

👍 1