



# spacerOS

The Future of Space Robotics



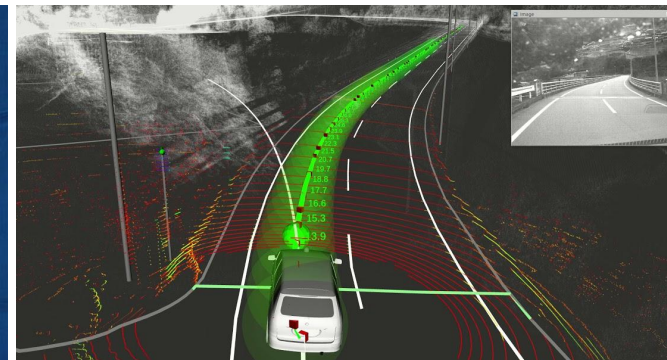
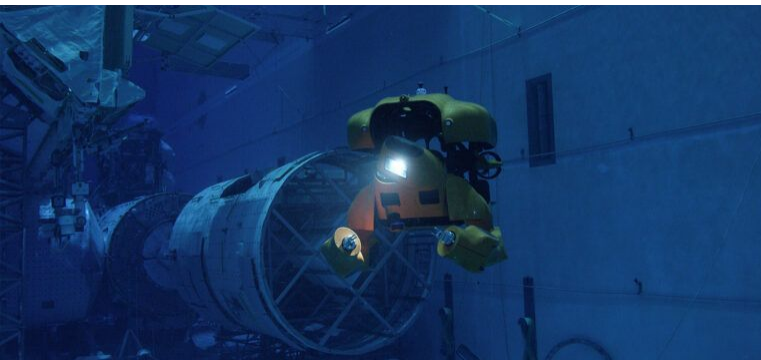
POZNAN UNIVERSITY OF TECHNOLOGY



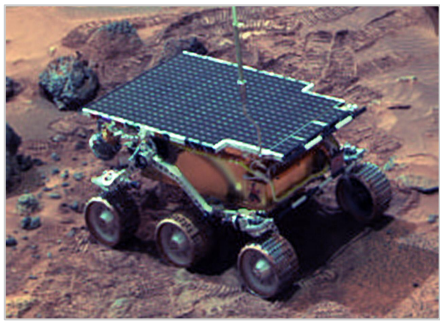
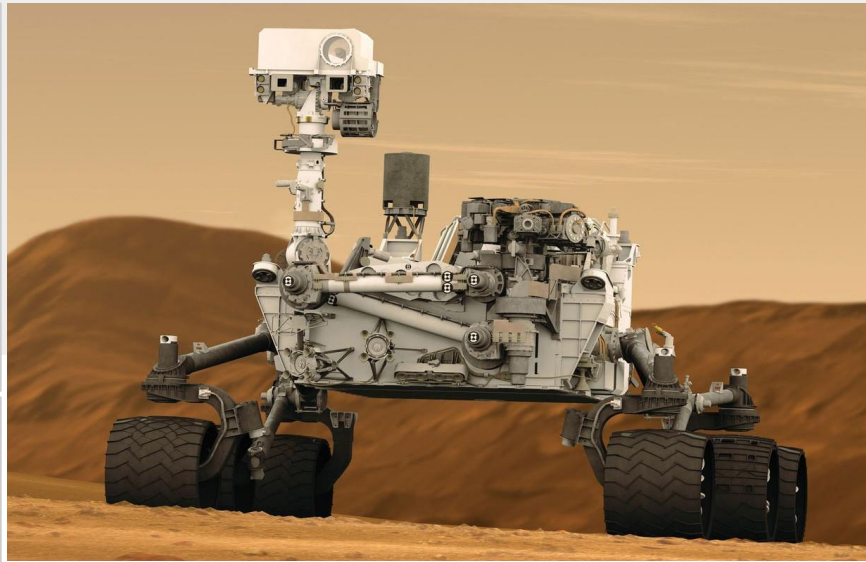
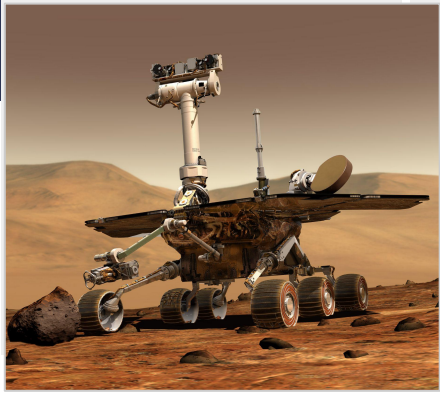
PICKNIK  
ROBOTICS



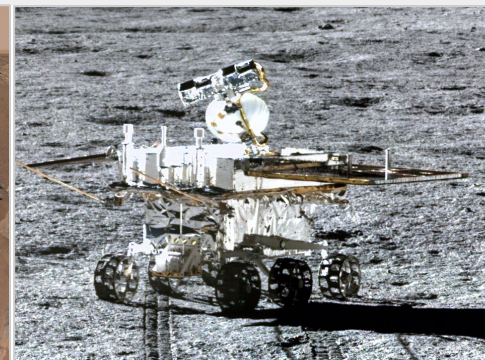
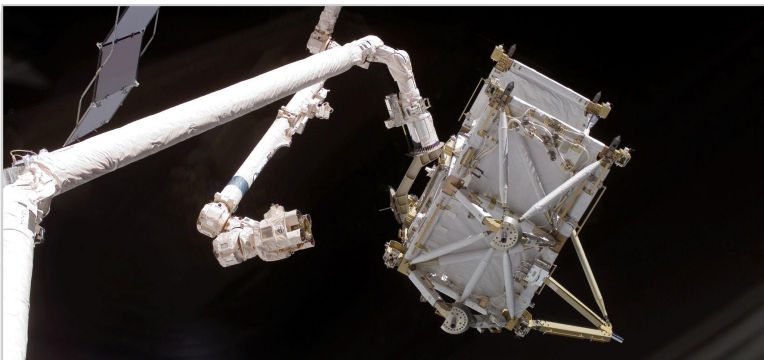
ROS robots are  
everywhere  
on Earth





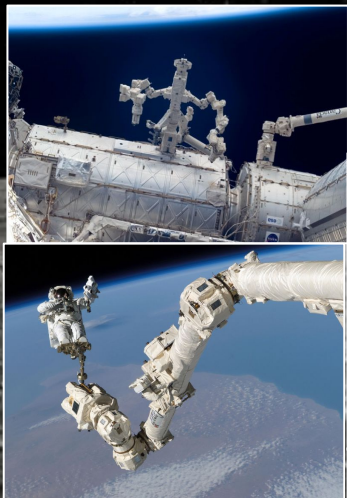


Robots are also in space

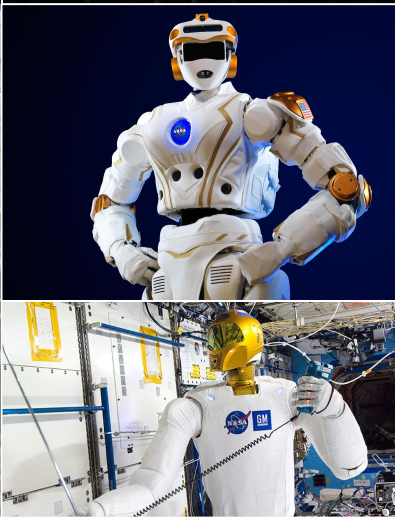




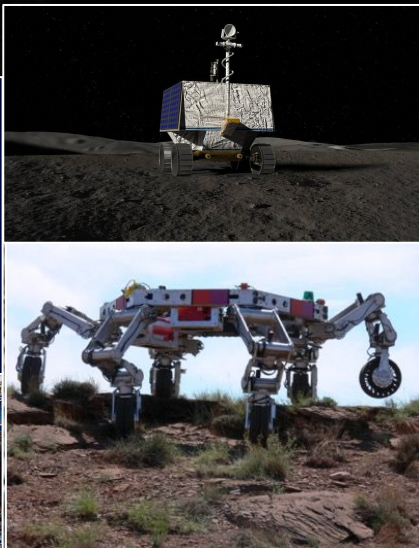
## Manipulators



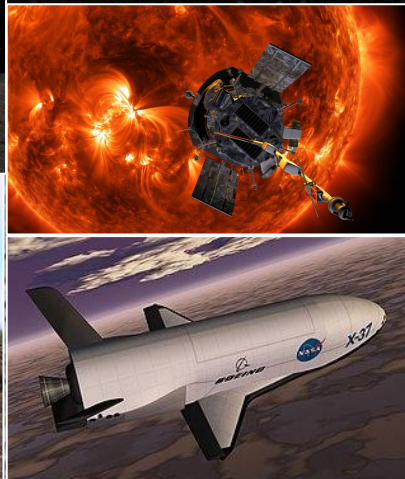
## Humanoid robots



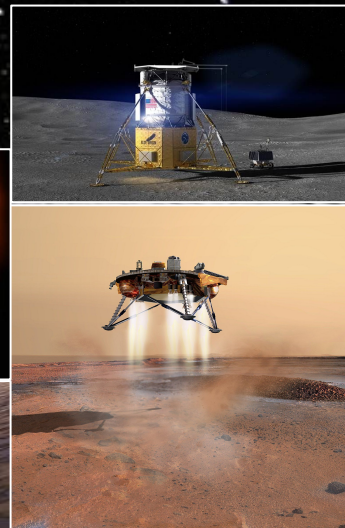
## Mobility systems



## Robotic spacecraft



## Robotic landers

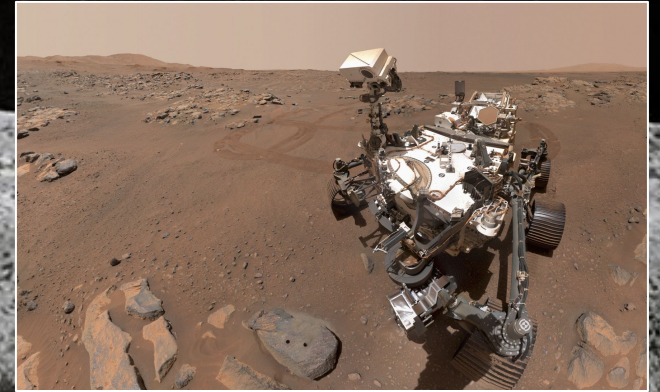




1970 – joystick



2020 – 2M lines of code



Increasing amount of software + cost of software development  
= *demand for reuse*



## The Demand for Reuse

The space community is already moving toward componentized, reusable, and open frameworks for flight software and mission control

- F' (F Prime)
- core Flight System (cFS)
- Yamcs
- OpenMCT
- ESROCOS
- TASTE

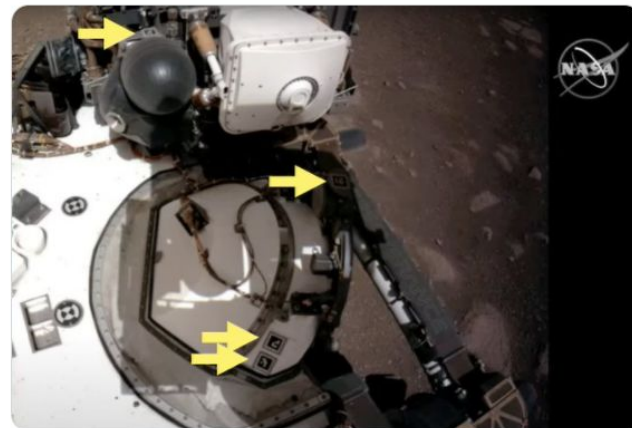
Also using smaller open source projects in flight

- AprilTag (visual fiducial system) used on Perseverance



Edwin Olson  
@edwinolson

Dear @NASAPersevere, what an amazing landing! Congratulations! My joy trebled when I saw that you have a number of AprilTag visual fiducials on board ([github.com/AprilRobotics/...](https://github.com/AprilRobotics/)). I pulled out the AprilTag iOS app, enabled the 16h5 tag family, and was able to read these tags!



6:44 AM · Feb 23, 2021 · Twitter Web App



# FPrime

**F' is a software framework for the rapid development and deployment of embedded systems and spaceflight applications**

- Originally developed at NASA's Jet Propulsion Laboratory, F Prime is **open-source** software that has been successfully deployed for several space applications
- **C++ framework** providing core capabilities like queues, threads, and operating system abstraction
- **Component architecture** with well-defined interfaces
- A **standard library** of flight-worthy components
- **Tools** for designing systems and automatically generating code from systems design
- **Testing tools** for unit and system-level testing
- It has been used for, but is not limited to, CubeSats, SmallSats, instruments, and deployables



# Core Flight system (cFS)

The core Flight System (cFS) is a platform and project-independent reusable software framework and set of reusable software applications

- The most common spacecraft flight software framework
- Provides a dynamic runtime environment
- Layered software, component-based design

*“It is the combination of these key aspects that makes it **suitable for reuse** on any number of NASA flight projects and/or embedded software systems at a **significant cost savings.**”*

– NASA.gov

# ROS-based robots have already been to space



2014: **Robonaut 2**



2019: **Astrobeer**



# NASA VIPER

Prospecting for lunar resources in permanently shadowed regions of the lunar south pole



- **ROS** used in ground software systems
- **Gazebo** simulation used in mission development, testing, planning, operator training, etc.
- Other open source software
  - cFS/ROS bridge
  - Yamcs
  - OpenMCT
- NASA requires software used in **flight missions** to be space qualified



What we need: A version of ROS for space applications!







## Space ROS

An open-source space robotics framework for developing flight-quality robotic and autonomous space systems

A wide-angle, fisheye photograph of a rover on a reddish-brown, rocky planetary surface. The rover is positioned in the center of the frame, facing away from the viewer. The terrain is rugged and uneven, with various sized rocks and craters. The sky is a pale, hazy orange, suggesting a thin atmosphere. The overall scene is desolate and arid.

## Our Goal

Ease the adoption of the popular ROS framework into space robotics systems



A photograph of a Space Shuttle External Tank (ET) and Solid Rocket Boosters (SRBs) in space. The ET is the large, white, cylindrical structure in the center, and the SRBs are the two large, white, cylindrical structures on either side. The background is the blackness of space, with some faint stars visible. The text is overlaid on the image.

## Certification-Ready

Provide software and artifacts that are aligned with aerospace standards

An aerial photograph of a desert canyon, showing layered rock formations in shades of brown, tan, and grey. The canyon walls are steep and show distinct horizontal strata. The floor of the canyon is a mix of sand and small rocks. The lighting creates strong shadows, highlighting the rugged terrain.

## Open Source and Open Community

Bring the benefits of ROS to  
space robotics





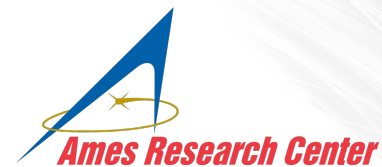
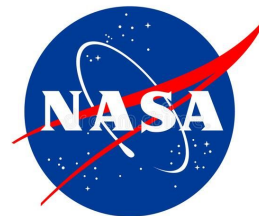
## A space-certifiable and reusable robotics framework

- Support certification to flight software standards, like DO-178C and NASA's NPR7150.2
- Provide artifacts to allow space flight projects to gain a head start on their certification efforts
- Aligned with NASA so that it can (eventually) be adopted for Class A missions
- Enable rapid development of new robotic capabilities
- Facilitate reuse across missions, reducing development effort and costs
- Based on open community, frameworks, and standards

## Genesis of Space ROS

### Blue Origin kick-started the Space ROS project

This project is funded in part by the Announcement of Collaboration Opportunity (ACO) program within NASA's Space Technology Mission Directorate, and Blue Origin Advanced Development Programs





# What is Space ROS?

Space ROS 2022

## Foundation

- Builds
- Releases
- Continuous Integration
- Maintenance
- Package subset
- Docker images

## Tools and Processes

- Requirements tools and processes for traceability and analysis
- Code analysis tools with SARIF output
- Dashboard for issue navigation, visualization & dispositioning
- Development workflow
- Quality level(s)
- MC/DC testing

## Space-Specific Functionality

- Eventing & Telemetry Subsystem
- C++ PMR allocator
- Sample applications
- Simulation assets
- Embedded target(s)

# What is Space ROS?

Space ROS 2022

## Foundation

- Builds
- Releases
- Continuous Integration
- Maintenance
- Package subset
- Docker images

## Tools and Processes

- Requirements tools and processes for traceability and analysis
- Code analysis tools with SARIF output
- Dashboard for issue navigation, visualization & dispositioning
- Development workflow
- Quality level(s)
- MC/DC testing

## Space-Specific Functionality

- Eventing & Telemetry Subsystem
- C++ PMR allocator
- Sample applications
- Simulation assets
- Embedded target(s)



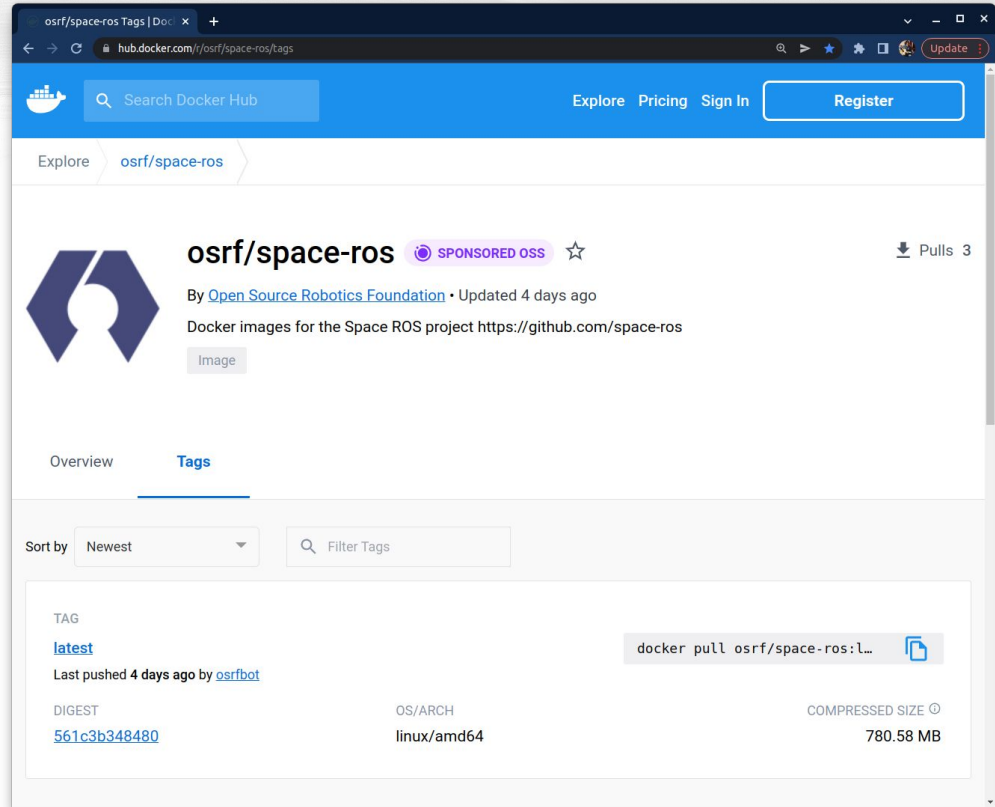
# Foundation

Keeping things running smoothly

Item	Description
<b>Space ROS GitHub Organization</b>	The Space ROS source code is hosted in a dedicated GitHub organization, <a href="https://github.com/Space-ROS">https://github.com/Space-ROS</a> .
<b>Automated Builds</b>	The CI system builds the Space ROS source code at a specified frequency and run all unit tests.
<b>Automated Build Output</b>	Space ROS developers can use the Jenkins UI to view the output of each automated build, which includes any issues discovered by the code analysis tools.
<b>Docker Images</b>	Also, via github Actions, we are also building the Space ROS docker image nightly: <a href="https://github.com/space-ros/docker/actions">https://github.com/space-ros/docker/actions</a> . This helps us to catch any regressions in the build and helps us to keep the Docker image in good shape for prospective users.
<b>Download of Space ROS Binaries</b>	Space ROS developers can download Space ROS images that result from each (successful) automated build.

# Foundation

Keeping things running smoothly



The screenshot shows the Docker Hub page for the `osrf/space-ros` repository. The page is titled "osrf/space-ros Tags" and is viewed in a browser window. The URL is `hub.docker.com/osrf/space-ros/tags`. The page features a blue header with a search bar, navigation links for "Explore", "Pricing", "Sign In", and a "Register" button. Below the header, there are breadcrumb links for "Explore" and "osrf/space-ros". The main content area displays the repository name "osrf/space-ros" with a "SPONSORED OSS" badge and a star icon. It also shows the pull count "Pulls 3". The repository is attributed to the "Open Source Robotics Foundation" and was updated "4 days ago". A description states "Docker images for the Space ROS project" with a link to the GitHub repository. There is an "Image" button below the description. Below the repository information, there are tabs for "Overview" and "Tags", with "Tags" being the active tab. A "Sort by" dropdown is set to "Newest", and there is a "Filter Tags" search box. The main content area shows a table of tags with columns for TAG, DIGEST, OS/ARCH, and COMPRESSED SIZE. The "latest" tag is highlighted, and a "docker pull" command is shown next to it. The digest for the "latest" tag is `561c3b348480`, the OS/ARCH is `linux/amd64`, and the compressed size is `780.58 MB`.

TAG	DIGEST	OS/ARCH	COMPRESSED SIZE
<a href="#">latest</a>	<a href="#">561c3b348480</a>	linux/amd64	780.58 MB



# Foundation

Keeping things running smoothly

Item	Description
<b>SARIF Viewing for all Code Analysis Tools</b>	In addition to generating JUnit XML to integrate with Jenkins, the CI system generates SARIF output for each of the code analysis tools used for Space ROS. This can be either native output from the code analysis tools, or output that is then converted to the SARIF format.
<b>Reporting of AUTOSAR C++ 14 Compliance Metrics</b>	The CI System generates AUTOSAR C++ 14 compliance metrics for each Space ROS package.
<b>Continuous Qualification System</b>	Working towards an automated end-to-end Continuous Qualification System that collates all of the reports and artifacts.
<b>Consistent Builds: CI and Local Environment</b>	Using Earthly to provide a uniform environment between CI and local developer environments.

# What is Space ROS?

Space ROS 2022

## Foundation

- Builds
- Releases
- Continuous Integration
- Maintenance
- Package subset
- Docker images
- Embedded target(s)

## Tools and Processes

- Requirements tools and processes for traceability and analysis
- Code analysis tools with SARIF output
- Dashboard for issue navigation, visualization & dispositioning
- Development workflow
- Quality level(s)
- MC/DC testing

## Space-Specific Functionality

- Eventing & Telemetry Subsystem
- C++ PMR allocator
- Sample applications
- Simulation assets
- Embedded target(s)



# Tools and Processes

Item	Description
<b>Code Conformance Updates</b>	Address issues identified by static analyzers (ongoing). Upstream changes.
<b>Space ROS requirements process</b>	Define a requirements process and associated tools for Space ROS.
<b>Integrate Requirements Analysis Tool</b>	Check requirements for consistency, conflicts, etc.
<b>Integrate MC/DC Tool</b>	Enable Modified Condition/Decision Coverage analysis. Required by DO-178C
<b>Analyze ETS Requirements</b>	Evaluate the requirements for the Eventing and Telemetry subsystem in order to prove out the methodology.
<b>Requirements for an existing Space ROS package</b>	Back-port requirements for a package, such as rcutils, and documentation on the process.
<b>Space ROS Quality Levels</b>	Define the Space ROS quality levels.

# Requirements management in **aerospace**

More than checklists

- Typically managed using a strict process and proprietary tools
  - Process is often according to some accepted standard, e.g. DO-178C
- Requirements must be complete - no software without requirements - and highly detailed
- Multiple levels of requirements - from abstract needs to detailed behaviour timings
- Traceability is essential - source to requirement to implementation and verification, and back again
- Requirements ultimately are used to support a certification process

# Requirements management in **open source software**

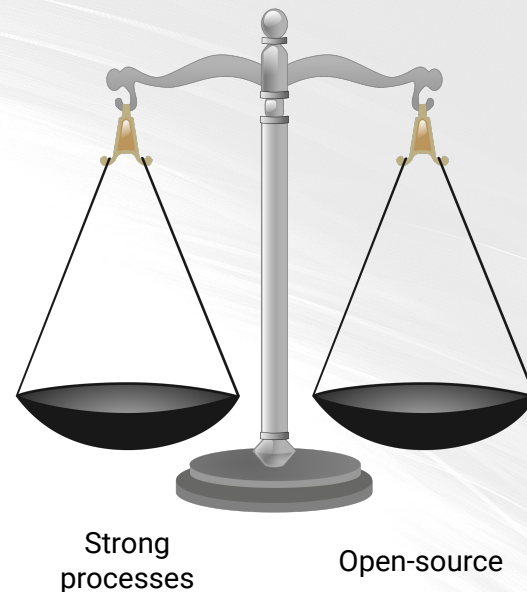
What requirements?

- Requirements are typically non-existent
- Any requirements that do exist are lightly managed (and easily get out-of-date)
- Heavy processes are shunned to avoid discouraging contributions



## Open requirements for Space ROS

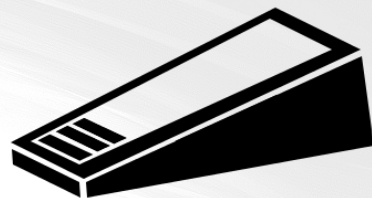
- Heavy-weight requirements process using expensive tools is inappropriate for an open-source project
- Need a process and tool(s) that won't discourage contributions
  - Contributors are unlikely to purchase expensive requirements management tools
  - Heavy-weight processes discourage drive-by contributors
- Must strike a balance between aerospace's need for strong processes and open-source's desire for ease-of-contributing



# Tools for open requirements management

## Doorstop

- Simple requirements management tool providing a command-line-and-text-editor based workflow
  - Add and edit requirements
  - Trace between requirements
  - Generate reports
- Based on YAML files stored in a versioned repository
  - Requirements are stored in a human-readable format
  - Easy to parse for additional automation tools
  - Requirements can be written in restricted natural language, e.g. EARS
- Open-source
  - Can be modified to meet our needs
  - Freely available to contributors
  - <https://doorstop.readthedocs.io/en/latest/>



# Tools for open requirements management

## FRET

- Graphical tool for creating and managing semi-formal and formal requirements
- Stores requirements in a database, with JSON import/export
- Requirements can be written in “FRETish”, which can contain linear temporal logic expressions
- Automatic model checking of requirements for consistency and conflicts
- Although freely available, the learning curve is steeper than Doorstop
- Automatic generation of safety monitor(s) from requirements expressed FRETish



**FRET** Projects [CREATE](#)

Total Projects	Total Requirements	Formalized Requirements	System Components	Requirement Size
5	133	87.97%	25	10594 bytes

**Hierarchical Cluster**

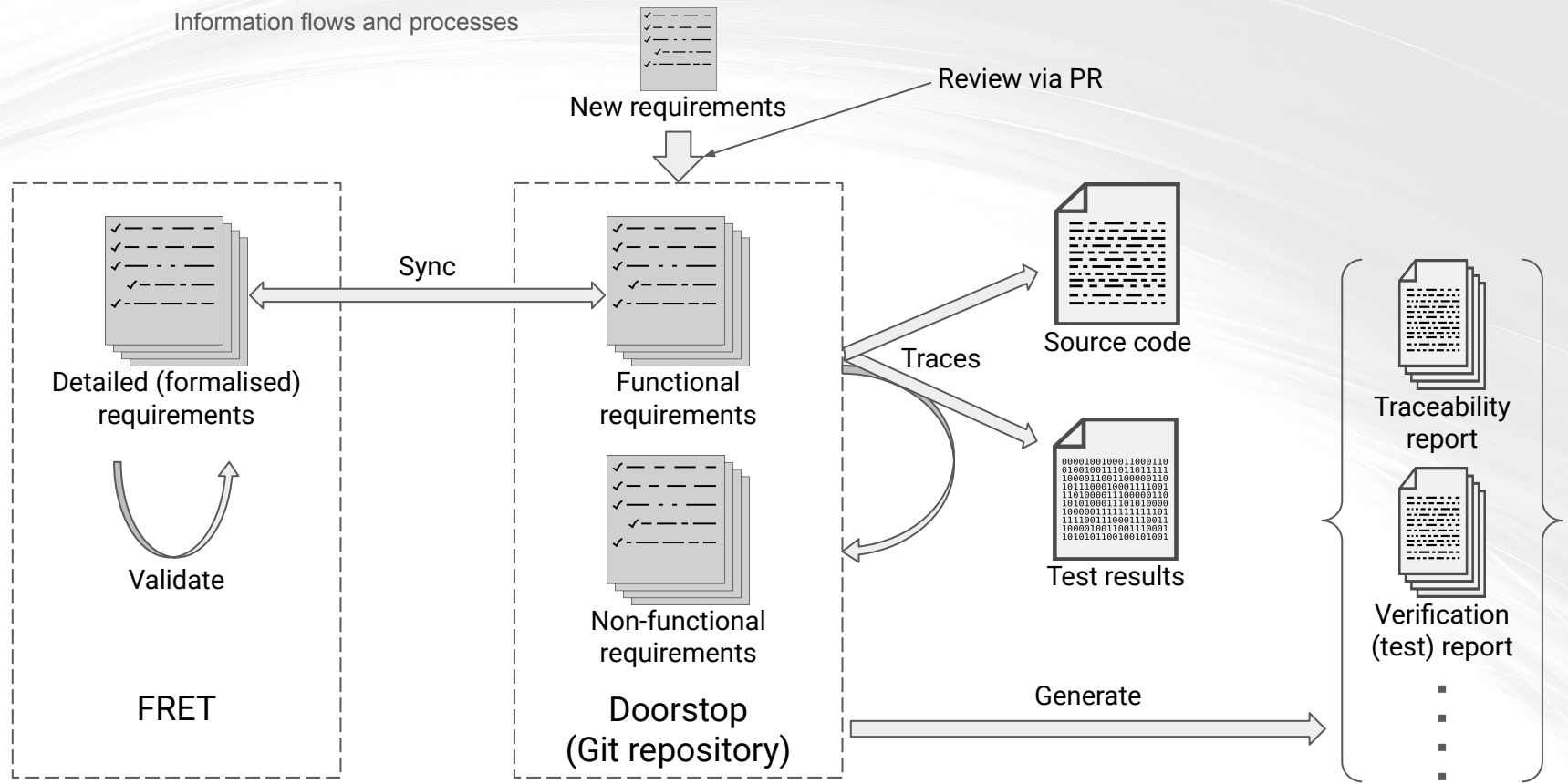
**Recent Activity**

- LM\_requirements FSM 006**  
FSM\_Autopilot shall always satisfy (state = ap\_maneuver\_state & standby & good) == STATE = ap\_standby\_state
- LM\_requirements EUL-001H**  
Euler shall always satisfy  $OCM321\_32 \in (-SinPhi * CosTheta + CosPhi * SinTheta * SinPsi)$
- LM\_requirements EUL-001B**  
Euler shall always satisfy  $OCM321\_12 \in CosTheta * SinPsi$
- LM\_requirements AP-003C**  
In roll\_hold mode RollAutopilot shall immediately satisfy  $abs\_roll\_angle \leq 30.0 \Rightarrow roll\_hold\_reference = 30.0 * sign(roll\_angle)$
- LM\_requirements AP-008B**  
In roll\_hold mode RollAutopilot shall always satisfy  $roll\_cmd = hdg\_hold\_mode\_cmd$
- LM\_requirements EUL-002B**  
Euler shall always satisfy  $R2\_21 \in V1\_1 * R\_21 + V1\_2 * R\_22 + R2\_3 * R\_23$
- LM\_requirements REG-003**  
Regulator shall always satisfy  $count\_yaw\_output\_exceeding\_50 \leq 100$
- LM\_requirements TSM-003a**  
TripleSignalMonitor shall always satisfy  $FC \neq 1 \Rightarrow set\_val =$



# Management of requirements in Space ROS

Information flows and processes



# Management of requirements in Space ROS

## Key points

- Doorstop used for:
  - High-level requirements
  - Non-functional requirements
  - Requirements traceability management
  - Artefact generation (e.g. traceability reports)
- FRET used for detailed functional requirements and consistency checks
- Requirements stored in Git (source of truth)
  - Pull requests provide a chance for requirements review
- Trace to implementation and tests via Git commit hashes

# Static Analysis

Meeting the needs of aerospace with open-source analysers

- Increase code quality, ease verification
- Space ROS provides a suite of static analyzers, including IKOS and Cobra from NASA
- Currently adding dynamic analysis: code coverage and MC/DC testing
- The static analysis tools generate SARIF output
  - Most by parsing output of the tool
  - Tools should eventually support SARIF directly; would allow for more detailed information in SARIF, such as logical location
- Filtering pass to remove (some) redundancy
  - Currently, removing identical issues
  - Would like to remove semantic equivalents
- The results are made available to the Space ROS Dashboard
  - An archive format that contains analyzer output, filtered output, and metadata



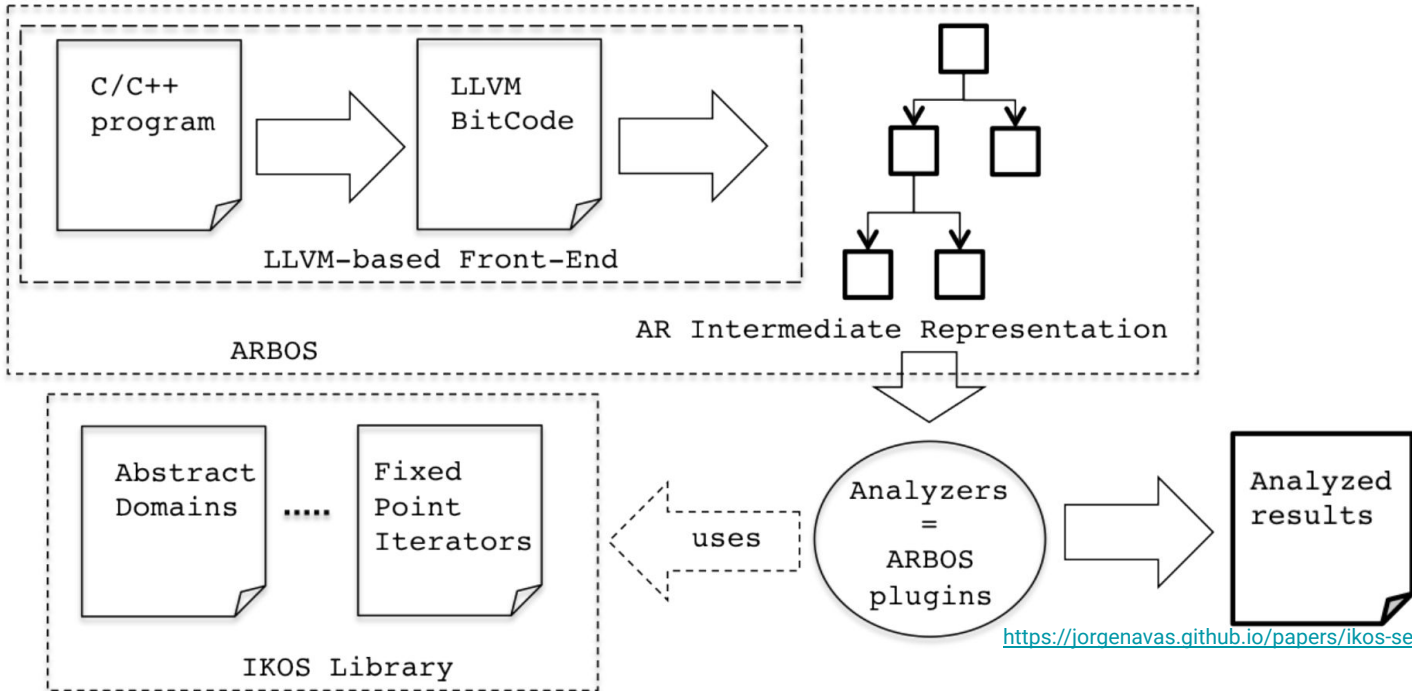
# IKOS (Inference Kernel for Open Static Analyzers)

Application of formal methods to support certification

- The RTCA standard DO-178C includes an extension, DO-333, that describes how developers can use static analysis in certification
  - DO-333 provides guidance on how **formal (mathematical) methods** may be used for the purpose of producing verification evidence suitable for use in certification
  - DO-333 lists **Abstract Interpretation** as a possibility to obtain certification credits
- IKOS is a static analysis framework, based on the Theory of Abstract Interpretation
  - Used to develop static analyses that are both precise and scalable
  - Makes it accessible to a larger class of static analysis developers
  - Separates concerns such as code parsing, model development, abstract domain management, results management, and analysis strategy
- References
  - <https://jorgenavas.github.io/papers/ikos-sefm14.pdf>
  - <https://github.com/NASA-SW-VnV/ikos>

# IKOS (Inference Kernel for Open Static Analyzers)

The IKOS framework architecture



# Cobra (code browser and analyzer)

An extensible, interactive tool for the analysis of C/C++ code

- A static analysis capability that works well for large code bases
- Fast analysis of general code patterns, common coding flaws, or coding rule compliance
  - Performs lexical analysis to generate a stream of language-level tokens
  - Stores the key information of source code in an extremely simple data structure
- Can be used in one of three modes
  - As an **interactive query engine** to match patterns with a simple query language
  - Execute **inline Cobra programs** that can contain arbitrary branching and iteration over the token stream to identify more complex types of patterns
  - As an infrastructure for building more elaborate **standalone checkers** that are compiled separately and linked with the Cobra code that builds the central data structure
- References
  - <https://software.nasa.gov/software/NPO-50050-1>
  - <https://github.com/nimble-code/Cobra>



# Cobra (code browser and analyzer)

An extensible, interactive tool for the analysis of C/C++ code

```
spaceros-user@ba0b59ced39b:~$ ament_cobra --help
usage: ament_cobra [-h] [--include_dirs [INCLUDE_DIRS [INCLUDE_DIRS ...]]] [--exclude [EXCLUDE [EXCLUDE ...]]] [--ruleset RULESET] [--compile_cmds COMPILE_CMDS]
                  [--xunit-file XUNIT_FILE] [--sarif-file SARIF_FILE] [--cobra-version] [--verbose]
                  [paths [paths ...]]
```

Analyze source code using the cobra static analyzer.

positional arguments:

paths Files and/or directories to be checked. Directories are searched recursively for files ending in one of '.c', '.cc', '.cpp', '.cxx'. (default: ['.'])

optional arguments:

-h, --help show this help message and exit

--include\_dirs [INCLUDE\_DIRS [INCLUDE\_DIRS ...]] Include directories for C/C++ files being checked. Each directory is passed to cobra as '-I<include\_dir>' (default: None)

--exclude [EXCLUDE [EXCLUDE ...]] Exclude C/C++ files from being checked. (default: [])

--ruleset RULESET The cobra rule set to use to analyze the code: basic, cwe, p10, jpl, misra2012, C++/autosar. (default: basic)

--compile\_cmds COMPILE\_CMDS The compile\_commands.json file from which to gather preprocessor directives. This option will take precedence over the --include\_dirs options and any directories specified using --include\_dirs will be ignored. Instead, ament\_cobra will gather all preprocessor options from the compile\_commands.json file. (default: None)

--xunit-file XUNIT\_FILE Generate a xunit compliant XML file (default: None)

--sarif-file SARIF\_FILE Generate a SARIF file (default: None)

--cobra-version Get the cobra version, print it, and then exit (default: False)

--verbose Display verbose output (default: False)

```
spaceros-user@ba0b59ced39b:~$
```

# SARIF (Static Analysis Results Interchange Format)

Unification of static analysis results

- A JSON-based exchange format for the output of static analysis tool
- Used by IDEs, code analysis tools, continuous integration systems, etc.
- SARIF output by all Space ROS static analyzers

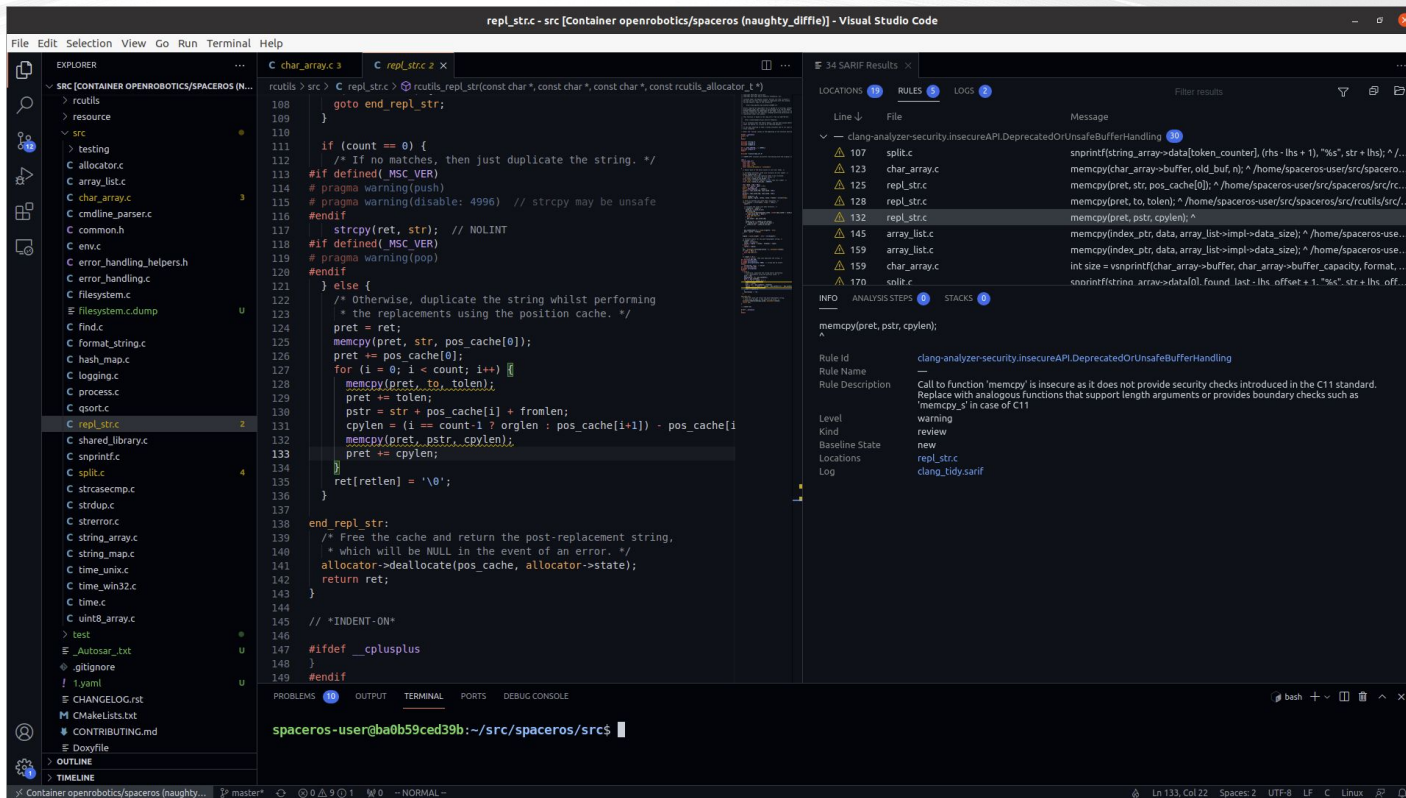
```

1  {
2    "version": "2.1.0",
3    "$schema": "http://json.schemastore.org/sarif-2.1.0-rtm.5",
4    "properties": {
5      "comment": "clang-tidy output converted to SARIF by ament_clang_tidy"
6    },
7    "runs": [
8      {
9        "tool": {
10         "driver": {
11           "name": "clang-tidy",
12           "version": "10.0.0",
13           "informationUri": "https://clang.llvm.org/extra/clang-tidy/",
14           "rules": []
15         }
16       },
17       "artifacts": [],
18       "results": []
19     }
20   ]
21 }
22

```

# VSCoDe SARIF plugin

Making static analysis results visible





# Extending the VSCode SARIF plugin

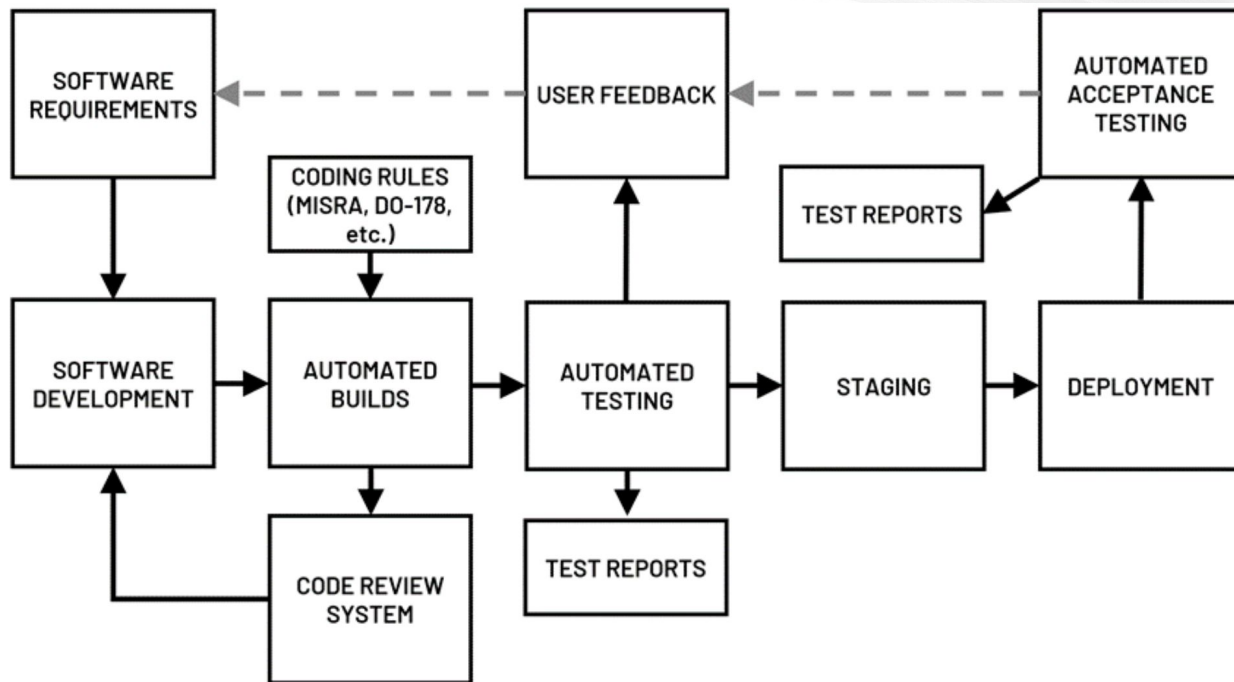
Making static analysis results visible

- Insight into static analysis, code coverage, build status, issue burndown, etc.
- A starting point for the open source community to extend and improve
- Interface to build, test, using Earthly (same as CI)
- Integrate with external dispositioning systems
- Plugin available on the VSCode Marketplace



# Static analysis process from CI to results

Support software quality and compliance activities



# What is Space ROS?

Space ROS 2022

## Foundation

- Builds
- Releases
- Continuous Integration
- Maintenance
- Package subset
- Docker images
- Embedded target(s)

## Tools and Processes

- Requirements tools and processes for traceability and analysis
- Code analysis tools with SARIF output
- Dashboard for issue navigation, visualization & dispositioning
- Development workflow
- Quality level(s)
- MC/DC testing

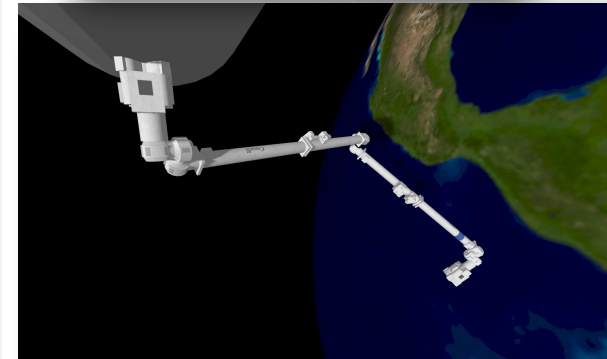
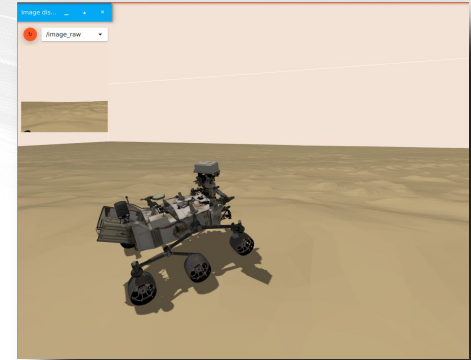
## Space-Specific Functionality

- Eventing & Telemetry Subsystem
- PMR allocator
- Sample applications
- Simulation assets



# Space-Specific Functionality

Item	Description
<b>cFS/ROS 2 Bridge</b>	TRAC Labs is working on " <a href="#">The BRASH Integration Toolkit for ROS2 and Flight Software Interoperability.</a> "
<b>Eventing and Telemetry Subsystem (ETS)</b>	The Events and Telemetry System provides event reporting functionality. It is used to instrument the software that executes on the spacecraft, to retain events for later reporting, and to perform the reporting.
<b>Custom Memory Allocators</b>	Space ROS applications can make use of a user-supplied allocator. Provide a sample application and sample allocator that demonstrates the use of a user-supplied allocator. Provide documentation that describes how to use a user-supplied allocator.
<b>Navigation and Manipulation Demo Apps</b>	Incorporate navigation (Curiosity Rover) and manipulation (Candarm) demo applications.
<b>Enable RTOS Build</b>	Build Space ROS for the <a href="#">RTEMS</a> embedded operating system, running on Qemu.
<b>Simulation Assets</b>	Incorporate space-related simulation assets that can then be available for use by Space ROS code.



# Ongoing development

Space ROS 2023+

## Foundation

- Regular releases
- Space ROS website
- Space ROS documentation site

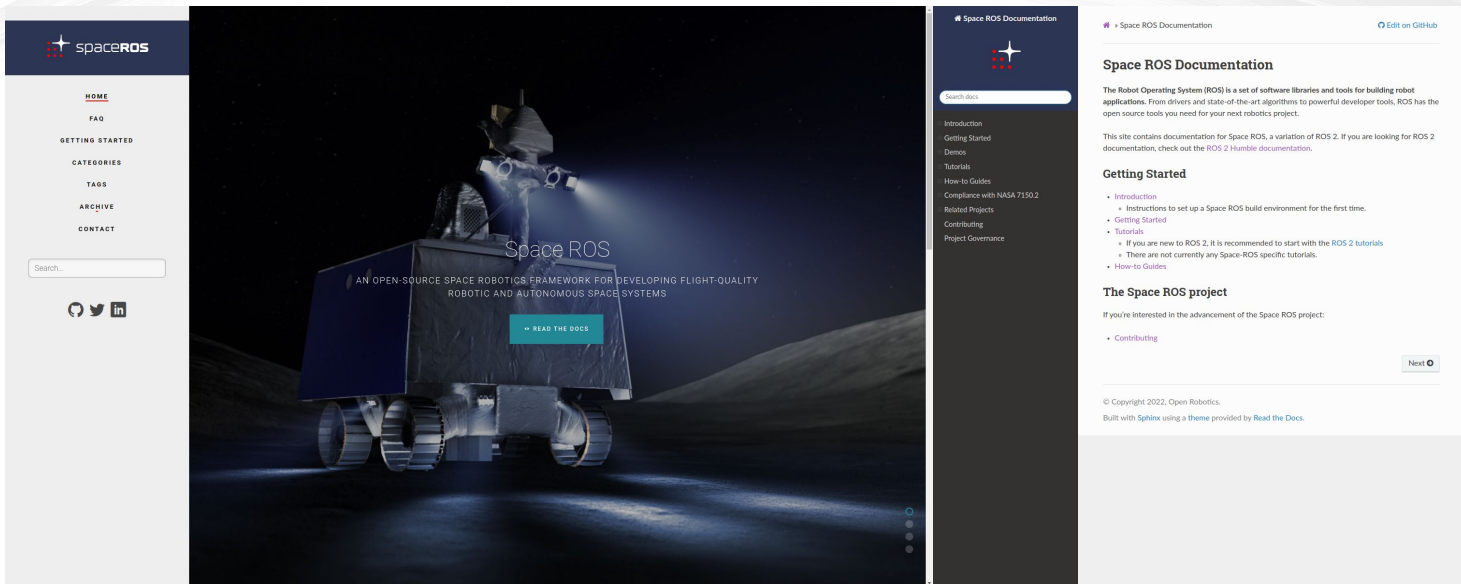
## Tools and Processes

- Continue Dashboard work based on user feedback
- Add auditing support, checklists, reports
- Continue to address issues identified by code analysis tools
- Back-port requirements
- Perform requirements analysis

## Space-Specific Functionality

- cFS/ROS 2 bridge
- Applications and demos

# Open processes and artefacts for community-driven validation



The screenshot displays the Space ROS website layout. On the left is a dark navigation sidebar with the spaceROS logo and a list of menu items: HOME, FAQ, GETTING STARTED, CATEGORIES, TAGS, ARCHIVE, and CONTACT. Below the menu is a search bar and social media icons for GitHub, Twitter, and LinkedIn. The main content area features a large image of a rover on a planetary surface. The text 'Space ROS' is centered over the image, with the subtitle 'AN OPEN-SOURCE SPACE ROBOTICS FRAMEWORK FOR DEVELOPING FLIGHT-QUALITY ROBOTIC AND AUTONOMOUS SPACE SYSTEMS' below it. A teal button labeled 'READ THE DOCS' is positioned at the bottom of the image. To the right of the main content is a dark sidebar with the title 'Space ROS Documentation' and a search bar. Below the search bar is a list of navigation links: Introduction, Getting Started, Demos, Tutorials, How-to Guides, Compliance with NASA 7150.2, Related Projects, Contributing, and Project Governance. The main content area on the right is white and contains the following text: 'Space ROS Documentation' with an 'Edit on GitHub' link, a paragraph describing ROS as a set of software libraries and tools for building robot applications, a note that the site contains documentation for Space ROS (a variation of ROS 2) and a link to ROS 2 Humble documentation, a 'Getting Started' section with a bulleted list of links (Introduction, Getting Started, Tutorials, How-to Guides), and a 'The Space ROS project' section with a link to 'Contributing'. At the bottom of the main content area is a 'Next' button. The footer contains copyright information: '© Copyright 2022, Open Robotics. Built with Sphinx using a theme provided by Read the Docs.'



# Open processes and artefacts for community-driven validation

- We're integrating open source tools and processes to help improve software quality
  - Requirements, code analysis, developer workflow, quality levels
  - This is done in the context of Space ROS, but could be useful to other domains
- We're working towards an end-of-year release of Space ROS
- We welcome your contributions and input
- <https://github.com/space-ros>

Krzysztof Walas: Poznan University of Technology, [krzysztof.walas@put.poznan.pl](mailto:krzysztof.walas@put.poznan.pl)  
IDEAS-NCBR, [krzysztof.walas@ideas-ncbr.pl](mailto:krzysztof.walas@ideas-ncbr.pl)



 spacEROs

