MaplessAI

Improving the Safety and Dependability of Open-source Autonomy

contact@mapless.ai

About us

Philipp Robbel

philipp@mapless.com in

- nuTonomy / Aptiv, Bosch HAD
- MIT PhD, Robotics
- Focus: AV Safety and Validation

Jeffrey Kane Johnson

jeff@mapless.com in

- Uber ATG, Apple SPG, Bosch HAD
- Indiana University PhD, C.S.
- Focus: Motion / Behavior planning

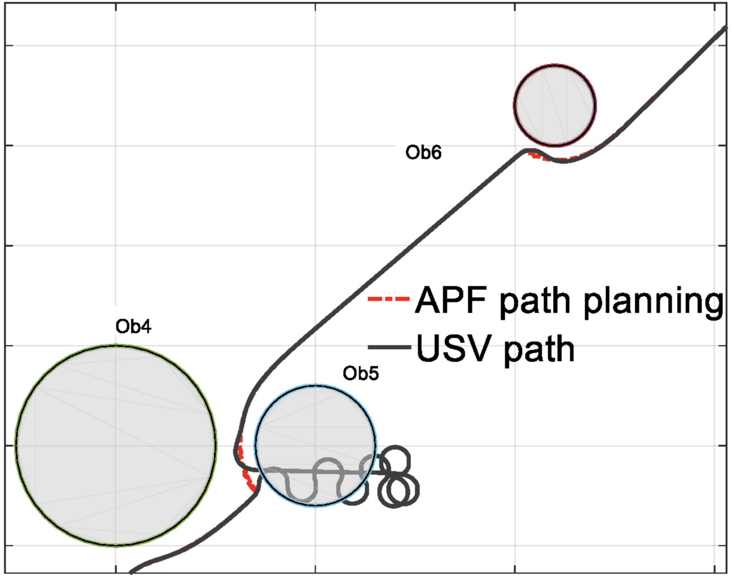
What does it mean to be safe and dependable?

- **Safety:** Free from unreasonable risk
- **Dependability:** The system consistently behaves as expected

Systems should be safe and dependable w.r.t. the **users** *and* **developers**

Safety can *change* the problem space

Find a smooth path in free space



Find a smooth in free space, ***iff one exists***

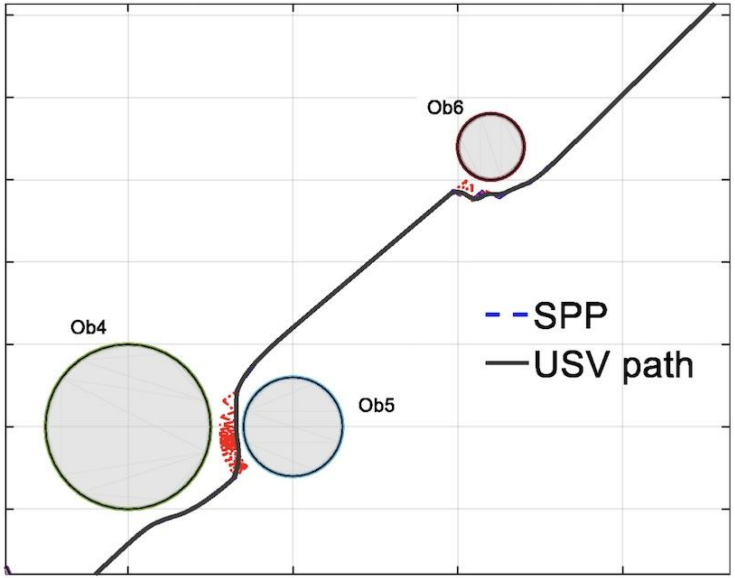
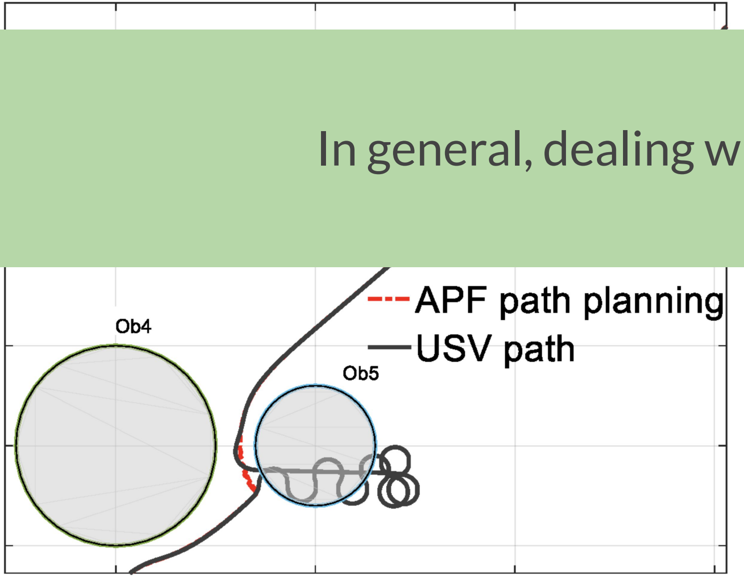


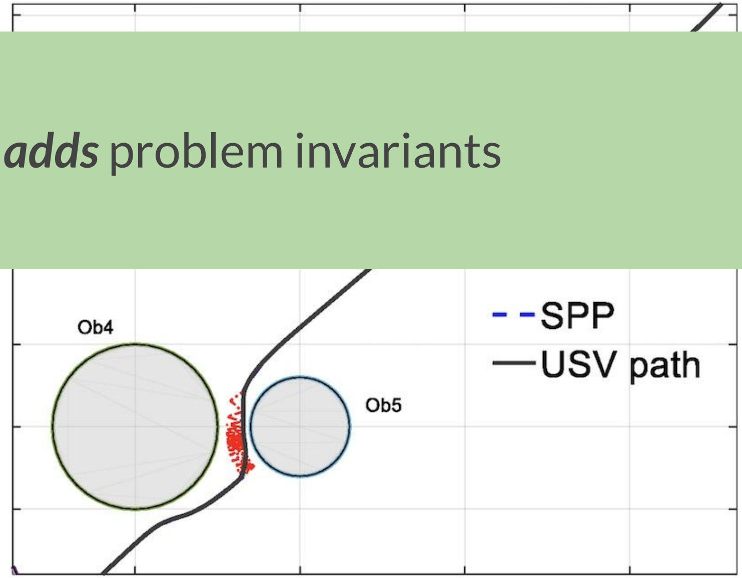
Image: *Second Path Planning for Unmanned Surface Vehicle Considering the Constraint of Motion Performance*, Fan, et al.

Safety can *change* the problem space

Find a smooth path in free space



Find a smooth in free space, ***iff one exists***



In general, dealing with safety ***adds*** problem invariants

Image: *Second Path Planning for Unmanned Surface Vehicle Considering the Constraint of Motion Performance,* Fan, et al.

Developing safer autonomy

1. Specify invariants and assumptions necessary for proper functioning

2. Implement such that invariants and assumptions are explicitly enforced

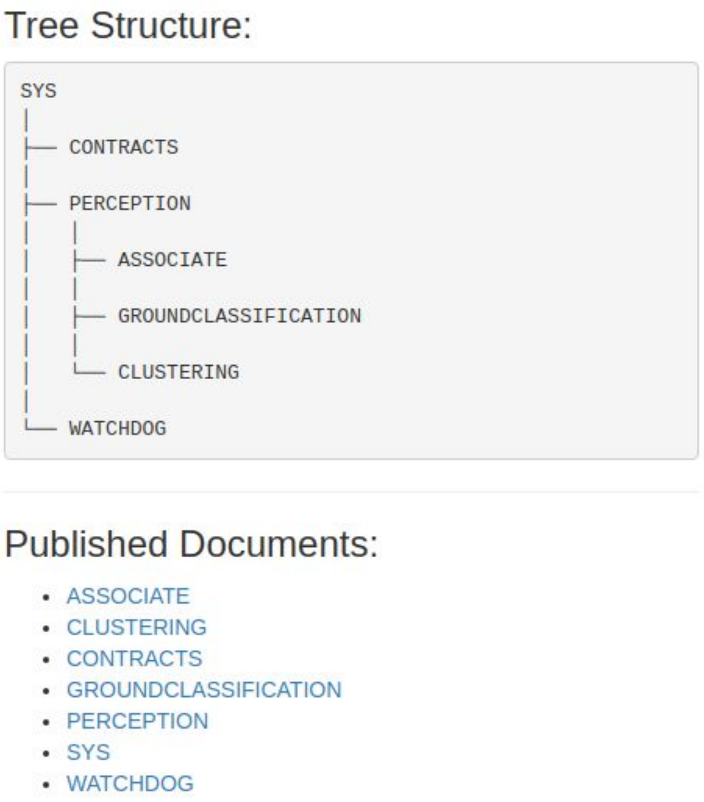
Invariant violations are ***significant*** sources of errors in autonomous systems

"Robustness Testing of Autonomy Software", *Hutchison, et al.*

Safety tools for open source

- Requirements specification
  - Based on Doorstop (https://doorstop.readthedocs.io/en/latest/)
  - Lives in repo, YAML-based
- Contract enforcement
  - Contract enforcement/handling mechanisms based on C++20 proposal
  - Contract types defined for re-usable checks
- Watchdog
  - Based on ROS 2 Lifecycle nodes
  - Two watchdog types and a composable heartbeat node
- All code contributed to:
  - Autoware.Auto: https://www.autoware.auto
  - ROS 2 Safety Working Group: https://github.com/ros-safety

Requirements specification for open source

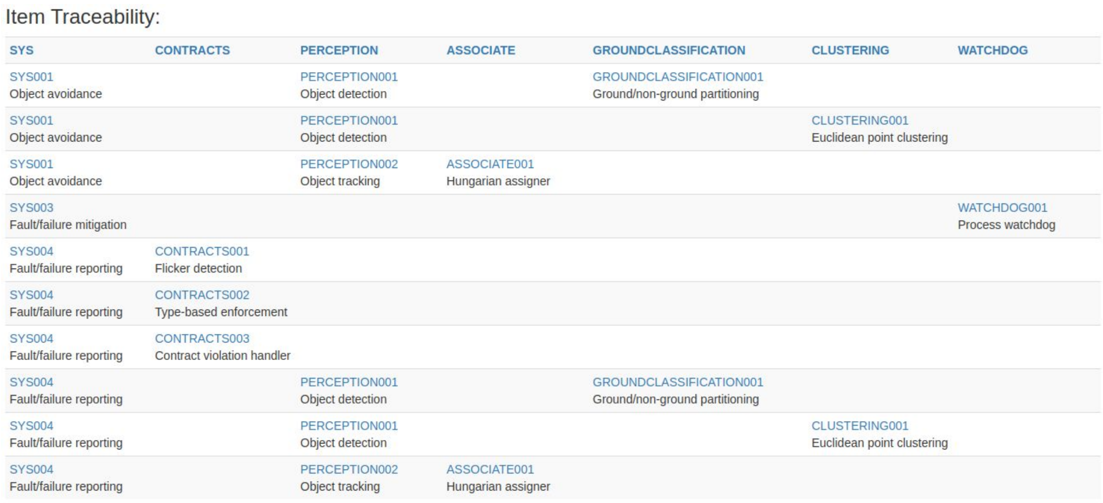


Tree Structure:

```
SYS
|
├── CONTRACTS
|
├── PERCEPTION
|   |
|   ├── ASSOCIATE
|   |
|   ├── GROUNDCLASSIFICATION
|   |
|   └── CLUSTERING
|
└── WATCHDOG
```

Published Documents:

- ASSOCIATE
- CLUSTERING
- CONTRACTS
- GROUNDCLASSIFICATION
- PERCEPTION
- SYS
- WATCHDOG

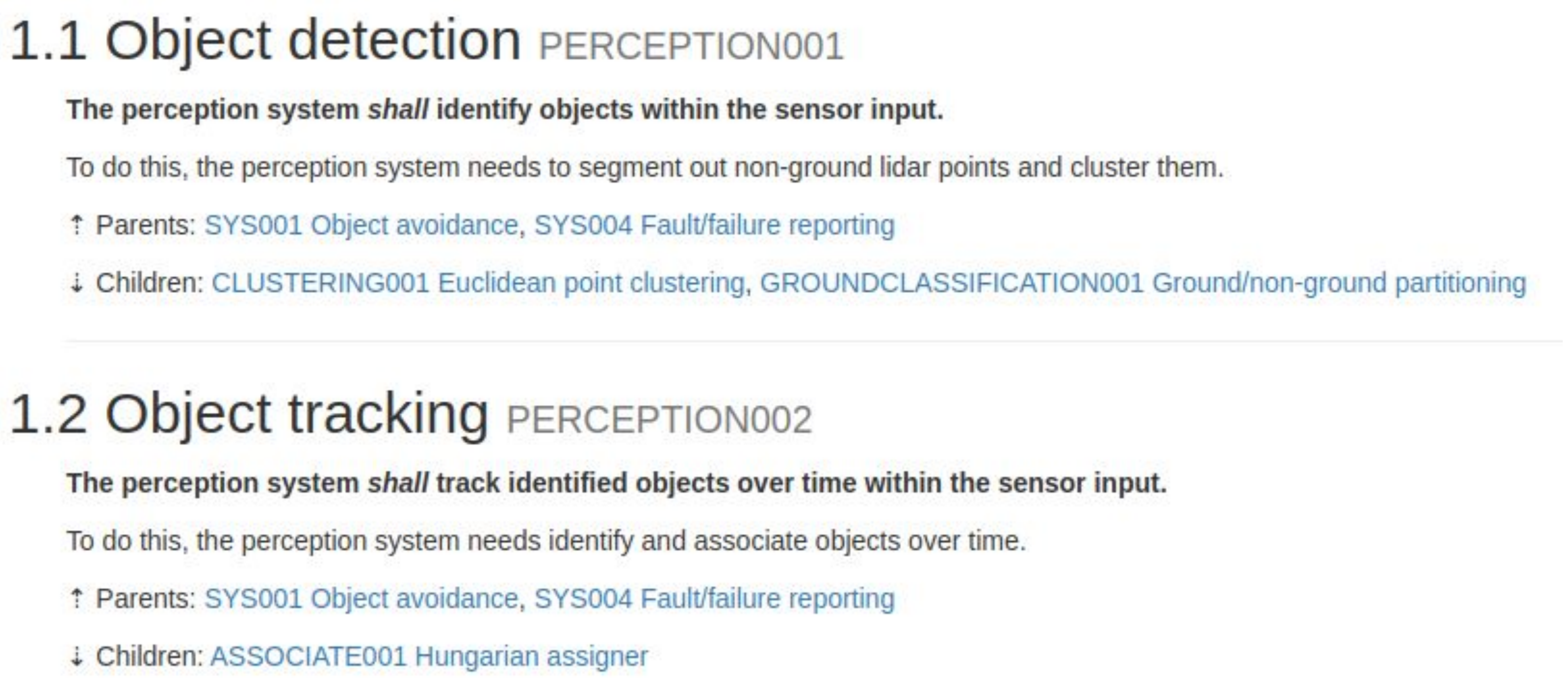


Item Traceability:

SYS	CONTRACTS	PERCEPTION	ASSOCIATE	GROUNDCLASSIFICATION	CLUSTERING	WATCHDOG
SYS001 Object avoidance		PERCEPTION001 Object detection		GROUNDCLASSIFICATION001 Ground/non-ground partitioning		
SYS001 Object avoidance		PERCEPTION001 Object detection			CLUSTERING001 Euclidean point clustering	
SYS001 Object avoidance		PERCEPTION002 Object tracking	ASSOCIATE001 Hungarian assigner			
SYS003 Fault/failure mitigation						WATCHDOG001 Process watchdog
SYS004 Fault/failure reporting	CONTRACTS001 Flicker detection					
SYS004 Fault/failure reporting	CONTRACTS002 Type-based enforcement					
SYS004 Fault/failure reporting	CONTRACTS003 Contract violation handler					
SYS004 Fault/failure reporting		PERCEPTION001 Object detection		GROUNDCLASSIFICATION001 Ground/non-ground partitioning		
SYS004 Fault/failure reporting		PERCEPTION001 Object detection			CLUSTERING001 Euclidean point clustering	
SYS004 Fault/failure reporting		PERCEPTION002 Object tracking	ASSOCIATE001 Hungarian assigner			

contact@mapless.ai

Requirements specification for open source



1.1 Object detection PERCEPTION001

The perception system *shall* identify objects within the sensor input.

To do this, the perception system needs to segment out non-ground lidar points and cluster them.

↑ Parents: SYS001 Object avoidance, SYS004 Fault/failure reporting

↓ Children: CLUSTERING001 Euclidean point clustering, GROUNDCLASSIFICATION001 Ground/non-ground partitioning

1.2 Object tracking PERCEPTION002

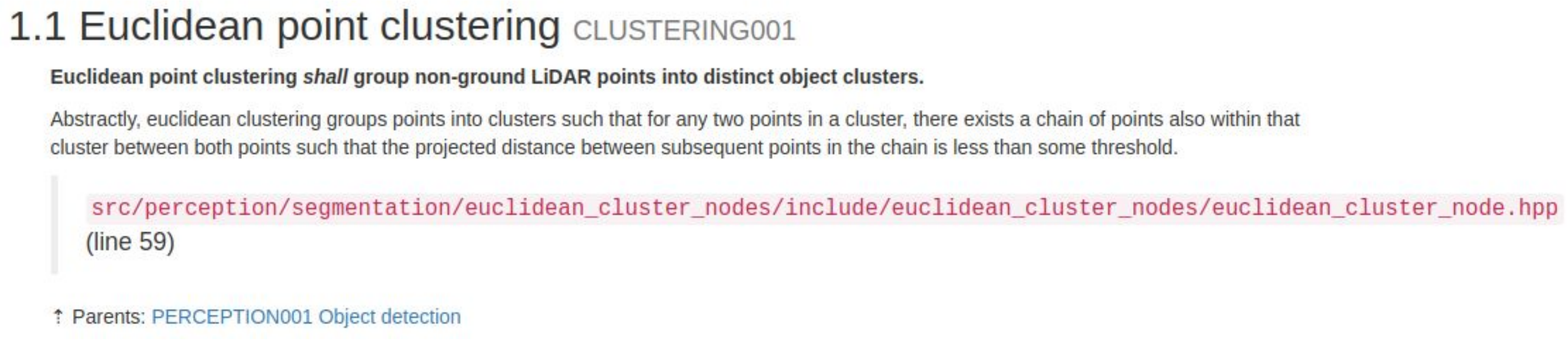
The perception system *shall* track identified objects over time within the sensor input.

To do this, the perception system needs identify and associate objects over time.

↑ Parents: SYS001 Object avoidance, SYS004 Fault/failure reporting

↓ Children: ASSOCIATE001 Hungarian assigner

Requirements specification for open source



1.1 Euclidean point clustering CLUSTERING001

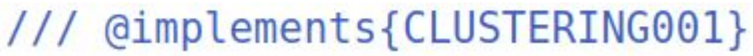
Euclidean point clustering *shall* group non-ground LiDAR points into distinct object clusters.

Abstractly, euclidean clustering groups points into clusters such that for any two points in a cluster, there exists a chain of points also within that cluster between both points such that the projected distance between subsequent points in the chain is less than some threshold.

> `src/perception/segmentation/euclidean_cluster_nodes/include/euclidean_cluster_nodes/euclidean_cluster_node.hpp` (line 59)

↑ Parents: PERCEPTION001 Object detection

Comment tag →



```
/// @implements{CLUSTERING001}
```

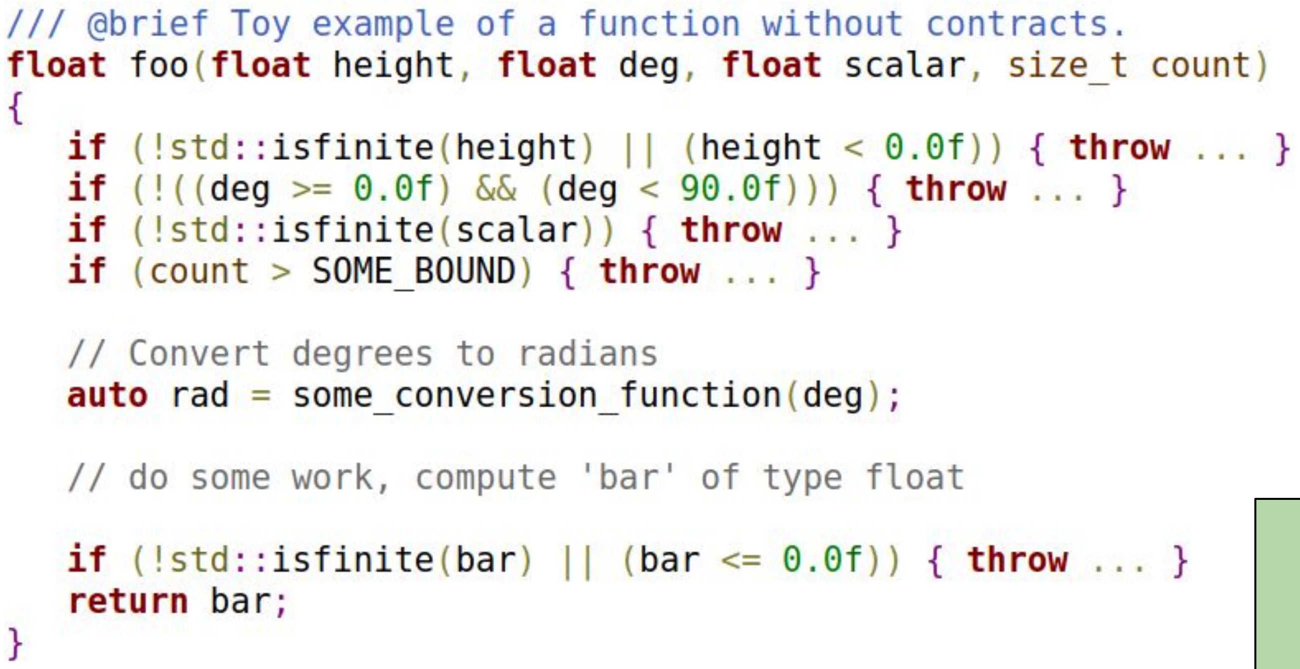
Contracts to verify requirements

- Explicitly defined pre-conditions, post-conditions, and invariants
  - Enforcement levels for range/plausibility checks as recommended by, e.g., ISO 262626

- Typically map well to requirements

- Many pluses:
  - Replace exceptions, remove invisible control flow
  - Treat errors as errors: unrecoverably bad program states

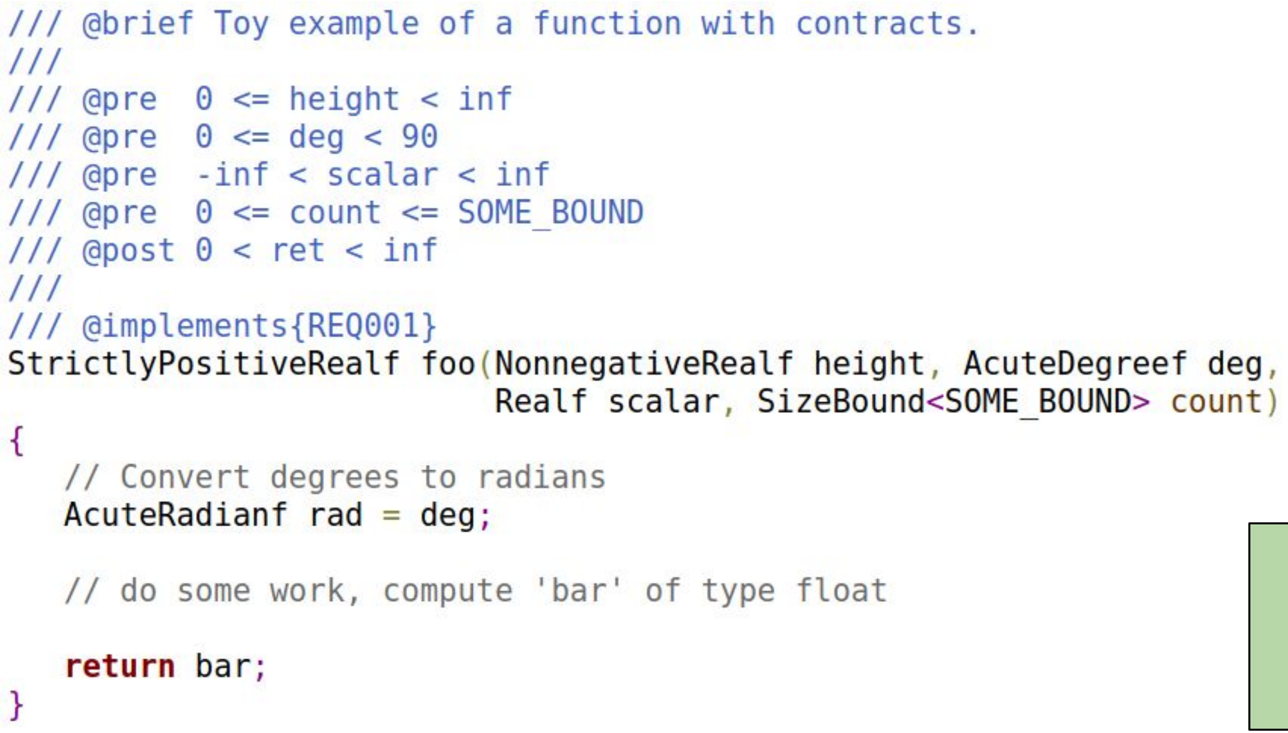
Contracts to verify requirements



```
/// @brief Toy example of a function without contracts.
float foo(float height, float deg, float scalar, size_t count)
{
   if (!std::isfinite(height) || (height < 0.0f)) { throw ... }
   if (!((deg >= 0.0f) && (deg < 90.0f))) { throw ... }
   if (!std::isfinite(scalar)) { throw ... }
   if (count > SOME_BOUND) { throw ... }

   // Convert degrees to radians
   auto rad = some_conversion_function(deg);

   // do some work, compute 'bar' of type float

   if (!std::isfinite(bar) || (bar <= 0.0f)) { throw ... }
   return bar;
}
```

Before contracts

Contracts to verify requirements



```
/// @brief Toy example of a function with contracts.
///
/// @pre  0 <= height < inf
/// @pre  0 <= deg < 90
/// @pre  -inf < scalar < inf
/// @pre  0 <= count <= SOME_BOUND
/// @post 0 < ret < inf
///
/// @implements{REQ001}
StrictlyPositiveRealf foo(NonnegativeRealf height, AcuteDegreef deg,
                          Realf scalar, SizeBound<SOME_BOUND> count)
{
   // Convert degrees to radians
   AcuteRadianf rad = deg;

   // do some work, compute 'bar' of type float

   return bar;
}
```

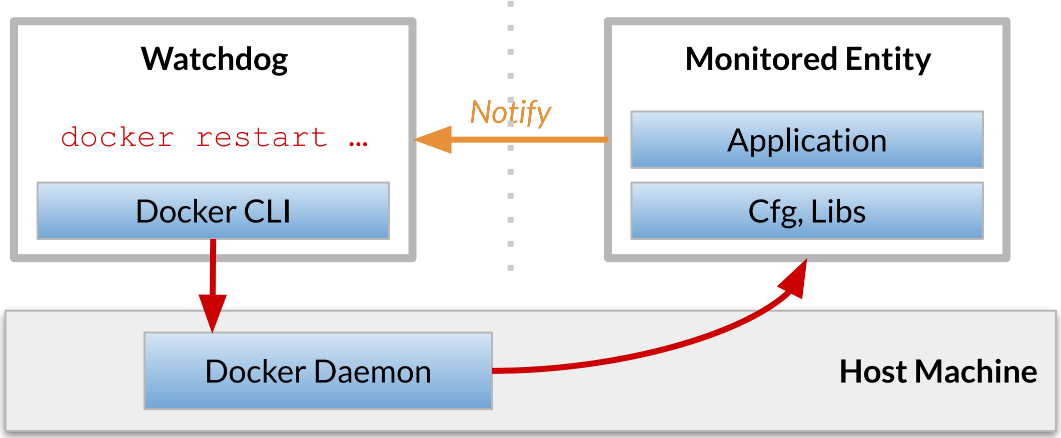
After contracts

Watchdog to respond to contract violations

- Expects a heartbeat at a specified frequency; otherwise declares a failure
  - Also windowed watchdog, support for checkpoints (program flow monitoring)
- Library based on DDS Quality of Service policies and ROS 2 lifecycle nodes
  - Uses DDS middleware features instead of re-implementing at higher layer
- Package includes a heartbeat node that can be added easily to an existing process via ROS 2 node composition
  - `ComposableNodeContainer` can do this dynamically at `ros2 launch` time

Process separation with Docker

- Safe applications must show freedom from interference between safety-critical and non-safety-critical code
    - Interferences can stem from CPU, memory, network bandwidth, disk, priority, *etc.*
- Docker can be used to increase process isolation and to configure host resource allocation per container (demo)



Autoware.Auto use case: Ray ground classifier

Specified and recorded requirements on:

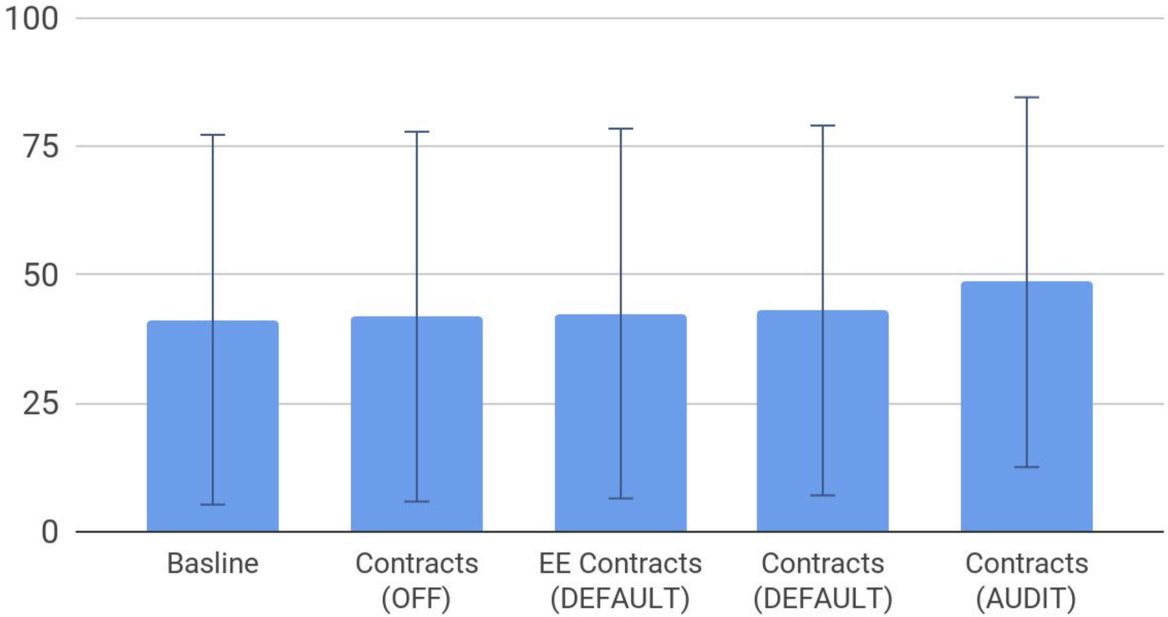
- Configuration data checking
- Assertion checking
- Runtime point cloud input/output checking

Implemented:

- Replace exceptions with contracts to perform invariant maintenance
- Issue 556: https://gitlab.com/autowarefoundation/autoware.auto/AutowareAuto/-/issues/556

Benchmark: Ray Ground Classifier Node

Mean callback time (ms) w/ 95% confidence interval



Where to find the code & documentation

- Watchdogs, requirements, & contracts:
  https://github.com/ros-safety

- Autoware.Auto integration (issue 556):
  https://gitlab.com/autowarefoundation/autoware.auto/AutowareAuto/-/issues/556

Reach out:
contact@mapless.ai