D4.11

Embedded Transform TF Library Software Release Y1

Grant agreement no.	780785
Project acronym	OFERA (micro-ROS)
Project full title	Open Framework for Embedded Robot Applications
Deliverable number	D4.11
Deliverable name	Embedded Transform TF Library – Software Release Y1
Date	December 2018
Dissemination level	public
Workpackage and task	4.4
Author	Ingo Lütkebohle (Bosch)
Contributors	
Keywords	micro-ROS, robotics, ROS, microcontrollers
Abstract	The micro-ROS tf2 filter provides a dynamic transform tree in an embedded device, while keeping resource use to a minimum based on an analysis of the spatial and temporal details actually necessary.

Contents

1 Overview to Results

This document provides links to the released software and documentation for deliverable D4.11 *Embedded Transform TF Library Software Release Y1* of the Task 4.4 *Embedded transform (TF) library*.

As an entry-point to all software and documentation, we created a dedicated webpage on the micro-ROS website: https://microros.github.io/embedded_tf/

The annex includes a copy of this webpage and copies of the major documentation files of this software release.

2 Links to Software Repositories

The tf2 filter node are provided in a micro-ROS fork of the ROS *geometry2* package:

- Git repository: https://github.com/microROS/geometry2/
  Package name: tf2_filter
  Package path: ./tf2_filter
  Pull request: 760ef72

Markdown source of the microros.github.io/embedded_tf/ webpage:

- Git repository: https://github.com/microROS/microros.github.io
  Webpage path: ./embedded_tf/
  Major commits: 7153cef

3 Annex 1: Webpage on Embedded TF

Content of https://microros.github.io/embedded_tf/ from 17th December 2018.

3.1 Introduction and Goal

The TF transform graph, with its support for both a temporal history, and distributed information sources, has been a novel tool for robotics frameworks when it was released in 2008. Functionally, it is based in scene graph concepts known from computer graphics [Foote 2013], but these only rarely offer distribution, and did not offer temporal histories at all (mainly, because this is not needed for frame-based rendering applications like in computer graphics). Distributed scene graphs have become more widely available also in computer graphics. In robotics, work by de Laet et al. [De Laet et al. 2013] has extended transforms graphs to also contain twist (i.e., angular motion) information, and to provide more compile-time error checking. This is currently not integrated with distribution mechanisms, but could be used on a single system.

One persistent issue with transform graphs has been their resource use. ROS TF works through replicated copies of the entire transform tree at every node that uses it, and is implemented through unicast TCP connections between nodes. In systems with many dynamic parts, this has sometimes been called the `TF firehose`, because of the large stream of incoming messages.

micro-ROS will go beyond this state of the art by running the dynamic transform tree in an embedded device, while keeping resource use to a minimum based on an analysis of the spatial and temporal details actually necessary. Further, enabling real-time queries even in the face of concurrent updates through integration will be realized through integration with the microROS real-time executor. It is also planned to integrate the embedded TF will with the node lifecycle to achieve further power-savings

3.2 Requirements

Embedded TF requirements are documented at:

github.com/microROS/geometry2/blob/ros2/tf2_filter/docs/requirements.md.

3.3 Design

The Embedded TF design is documented at:

github.com/microROS/geometry2/blob/ros2/tf2_filter/docs/design.md.

3.4 Implementation of tf2_filter

Implementation of tf2_filter for ROS2 and micro-ROS can be found at:

github.com/microROS/geometry2/blob/ros2/tf2_filter/.

3.5 Roadmap

2018

- Static filter approach in the agent allowing to specify the parts of the kinematic chain that are relevant for an application component on micro-ROS.

2019

- Design and implement an embedded TF implementation which is integrated with the real-time executor in a way that removes the need for costly synchronization primitives.

2020

- Design and implement an API extension, as well as a reference implementation for custom transform representations that increase run-time efficiency.

3.6 Acknowledgments

This activity has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement n° 780785).

4 Annex 2: README.md from tf2_filter Package

Content of https://github.com/microROS/geometry2/blob/ros2/tf2_filter/README.md from 17th December 2018.

4.1 Introduction

Simple package for filtering a TF stream by link.

Subscribes to `/tf` and publishes to `/tf_filtered`, by default.

4.1.1 Synopsis

```
ros2 run tf2_filter tf2_filter_node base_link arm1
```

This will pass through all transfroms from `base_link` to `arm1`.

For chains with multiple links, simple give all links in order as arguments.

4.1.2 Example

You can run `tf2_filter_example_publisher.py` to give you an example /tf stream (with empty transforms…). It publishes two transforms. With the TF2Filter invocation above, only one will remain.

4.2 License

tf2_filter is open-sourced under the Apache-2.0 license. See the LICENSE file for details.

5 Annex 3: requirements.md from tf2_filter Package

Content of https://github.com/microROS/geometry2/blob/ros2/tf2_filter/docs/requirements.md from 17th December 2018.

5.1 Introduction

5.1.1 Overall Goal

The goal of this package is to reduce bandwidth between machines and CPU load on transform users.

5.1.2 Terms

- *Frame ID:* The unique identifier of a coordinate frame.
- *Transform:* Translation and rotation relating two coordinate frames.
- *Transform chain:* A series of transforms without breaks, to relate coordinate frames.
- *Transform user:* The node that makes use of transforms to realize functionality.
- *Filter:* A pair of frame_id's, the specifies a transform chain of interest for the user.

5.1.3 System Use Cases

1) A semi-autonomous drone swarm, where drones receive information about other drones
2) An autonomous domestic robot that does internal sensor-frame to base_link transformations, and also sends its map-pose, mainly for debugging

5.1.4 Assumptions

- The user of the filtered transform is not on the same machine as the generator. Otherwise there are better approaches to exchange the data.
- TF users do not actually need all data. Otherwise there's nothing to filter.
- Once a transform chain exists, its constituent links stay the same (simplicity)
- Latency is important (otherwise we would use the `buffer_server`)
- Bandwidth and/or CPU is scarce (otherwise we would send everything)

5.2 Requirements

5.2.1 Data

- As a user, I want the output to be TFMessage, so that I can simply remap /tf->/tf_filtered for the target node and everything else stays the same.
- As a CPU-limited user, I would like to receive a transform chain collapsed to a single transform so that I don't have to compute it myself.

5.2.2 Configuration

- As an integrator, I want to specify to be able to specify the target chain in the configuration, so that the target nodes don't have to be modified.

- As a developer, I also want to be able to specify the (or a) chain at runtime, so that I can specify it based on runtime information.
- As a performance engineer, I want one filter node to be able to process multiple transform chains so that the resource use associated with receiving and processing incoming data occurs only once.

6 Annex 4: design.md from tf2_filter Package

Content of github.com/microROS/geometry2/blob/ros2/tf2_filter/docs/design.md from 17th December 2018.

6.1 Design Notes

Structurally, both a standalone implementation as well as a filter node will be provided. The latter is ready-to-use, and can even be composed in, the former can be embedded into existing nodes (such as the agent) to reduce latency.

The processing itself is trivial: Message in, check frame_id and child_frame_id, filtered message out.

The only moderately difficult part is knowing which parent-child pairs to pass through. In the general case, this requires looking at the robot model. And there might be parts that are only resolved at runtime.

In the first instance, we simply avoid this by making it a configuration parameter. For many simple systems (which we are targeting), setting this will be straightforward.