Lab02 - Transformations and Client-Server

MTRN4231 - UNSW School of Mechanical and Manufacturing Engineering

Preliminaries

Watch the lectures, They are required for this course. This lab assumes a basic understanding of Python and computer vision. If unfamiliar please review MTRN3100 computer vision content.

Introduction

This lab introduces one of the most critical aspects of ROS2 — mainly the TF2 (Transformation 2) Library. Remember in 4230 having to manually compute transformation matrices for each link of the robotic arm? This library does that all for you and even more.

If you are stuck during this lab, refer to the demo packages in `'4231/4231_demo_packages'`.

Task 1 - Static Transformation

A static transformation does not change over time. Examples of a static transformation may include, a TCP to the arm wrist, or wheels to a drive base. For this lab task, we will place a static transformation between the Robotic arm and the Depth Camera.

- Navigate to `'/4231/lab2_workspace/src'` and create a C++ Package called `'Transformations'`.
- Create a new file called `'static_camera_transformation.cpp'` and paste the provided stub in it.
- Complete the file according to the parameters below.
- Fill in CMake and Package.xml.
- Compile your program and run the node.
- In a separate terminal open Rviz2 and add the TF view.

Static transformation parameters

The transformation stamped message used to broadcast the frame can be found Here. Additionally, a quaternion calculator can be found Here.

- header.frame_id = “map”
- child.frame_id = “camera_frame”
- x = 1.2, y = 0.0, z = 1.0
- Rotation - facing down at 45 degrees use the online quaternion calculator.

Required packages

Add these packages to your CMake and package.xml list.

- rclcpp

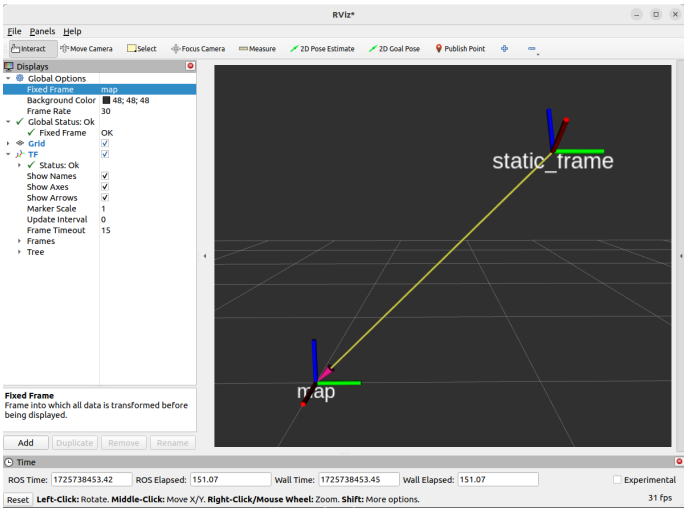

Figure 1: Example of static transformation displayed in Rviz. Note this example transformation is different to the lab task.

- tf2
- tf2_ros
- geometry_msgs

Task 2 - Dynamic Broadcaster

A static transformation does not change during the program's operation, while a dynamic transformation does. Dynamic transformations are often used for object tracking and movement. This task will move the transformation in a sin wave; however, later, you will approximate the distance to a target and calculate the associated transformation frame. The Ros2 Tutorial for TF2 can be found Here.

- Navigate to `'/4231/lab2_workspace/src/Transformations'`
- Create a new file called `'dynamic_transformation.cpp'` and paste the provided stub in it.
- Complete the file according to the parameters below, your dynamic transformation frame should publish at 10 Hz.
- Fill in CMake and Package.xml.
- Compile your program and run the node.
- In a separate terminal open Rviz2 and add the TF view.

Dynamic transformation parameters

- header.frame_id = "camera_frame"
- child.frame_id = "OOI"
- pos — x = 0.5 , y = 0.0 , z = sin(t)
- rot — x = 0.0 , y = 0.0 , z = 0.0 , w = 1.0

Task 2.1 - Useful View Frames Tool

Visualisation transformation frames can be hard, luckily there is a useful tool built into the library which listens to the transformation frames and generates an image showing all the frames and relations.

- Run the associated transformation nodes and verify that they display in rviz.
- In a new terminal window run `'ros2 run tf2_tools view_frames'`
- Open the resulting PDF and view the frame tree you have constructed.

Task 3 - Transformation Listener

To perform a lookup operation a transformation listener can be used. For instance, here we wish to know where the OOI is relative to the map transformation frame.

- Navigate to `'/4231/lab2_workspace/src/Transformations'`
- Create a new file called `'listener_transformation.cpp'` and paste the provided stub in it.
- Complete the file according to the parameters below.
- Print the resulting transformation lookup to the terminal.
- Fill in CMake and Package.xml.
- Compile your program and run the node.
- In a separate terminal open Rviz2 and add the TF view.

Listener parameters

- fromFrameRel = "map"
- toFrameRel = "OOI"

Task 4 - Visualisation

Visualisation is a useful tool to render and display an item in rviz. While you can display robot models and more intricate URDFs (lab5), for this lab we will visualise a simple ball.

- Navigate to `'/4231/lab2_workspace/src'` and create a C++ Package called `'Visualisation'`.
- Create a new file called `'OOI.cpp'`.
- Publish a red ball `'visualization_msgs'` at the OOI transformation frame at a frequency of 1Hz.
- Fill in CMake and Package.xml.
- Compile your program and run the node.
- In Rviz2 add the Marker `'Add -> By topic -> Marker'` you should see the marker displayed on the transformation frame.

NOTE: As the visualization is publishing at a lower frequency than the transformation the visualization marker will not track the transformation, rather it will lag. When working with robotics it is important to be aware of your operating frequency and ensure relevant systems are compatible.

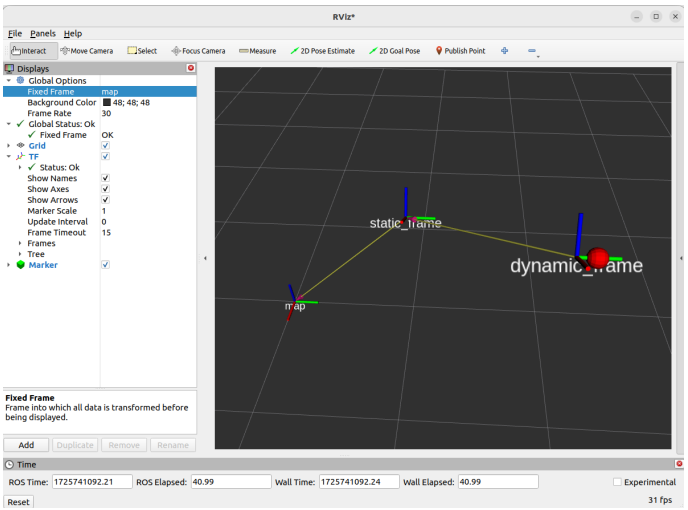

Figure 2: Example transformations with marker visualisation using the demo packages.

Task 5 - Client-Server

Hopefully, you are familiar with the publisher-subscriber relationship. This task aims to introduce a new concept of client-service relationship which enables bi-directional communication. A client sends a request to a server which then responds with a result. A publisher sends a message, but for client-server relationships, this is referred to as a service. Like messages, a custom service can be used.

Note: DO NOT refer to the humble tutorials for Client Service relationship examples, rather refer to the example in this lab task. The humble tutorial blocks and prevents other calls from occurring while waiting for a response. To avert this our example uses std::bind and a callback function rather than a threadblocking wait function. When programming with Ros NEVER use thread-blocking code, Ie Delays, While loops, etc....

- Read through and run the client service demo provided in the lab2_workspace.
- Add a new server that multiplies the request rather than adds it. use the same service.

Task 6 - Transformation service

As services can sort of be thought of as acting as functions, the Code that is used by multiple nodes should be a service call. An example of this may be the transformation lookup. Make a custom service call using the template below which performs a loop-up operation between two frames. Ensure you add your custom interface package to CMake and Package.xml. Documentation for custom messages and service calls can be found Here.

Demo template

```
string toFrame
string fromFrame
---
double x
double y
double z
```

Steps

- Run the associated transformation nodes and verify that they display in rviz.
- Run your transformation lookup node.
- Open RQT from a terminal that has sourced your custom interfaces package. If you do not source your interface package then rqt will not be able to see your custom interfaces.
- In RQT `'plugins -> Services -> Service Caller'`
- Refresh the service list and select your custom service. Set your fromFrame to map and toFrame to OOI.
- Call the server and view your result.