Lab 3 Report

Robotics Integration Group Project I

Yuwei ZHAO (23020036096)
Group #31 2025-11-26

Abstract

See Resources on github.com/RamessesN/Robotics_MIT.

1 Introduction

2 Procedure

2.1 Individual Work

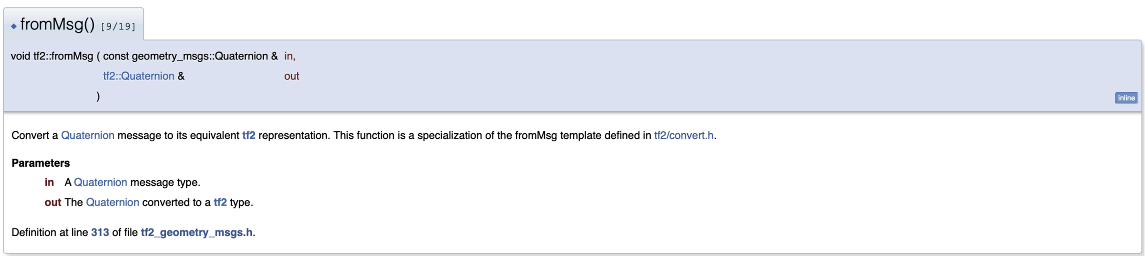
2.1.1 Transformations in Practice

1. **MESSAGE VS. TF**

   - **Assume we have an incoming `geometry_msgs::Quaternion quat_msg` that holds the pose of our robot. We need to save it in an already defined `tf2::Quaternion quat_tf` for further calculations. Write one line of C++ code to accomplish this task.**

     ```
     tf2::fromMsg(quat_msg, quat_tf);
     ```

     More specifically, we can find the official documentation of `fromMsg()` at this page:

     ◆ fromMsg() [9/19]

     void tf2::fromMsg (const geometry_msgs::Quaternion & in,
     tf2::Quaternion & out
)
     inline

     Convert a Quaternion message to its equivalent tf2 representation. This function is a specialization of the fromMsg template defined in tf2/convert.h.

     Parameters
     in A Quaternion message type.
     out The Quaternion converted to a tf2 type.

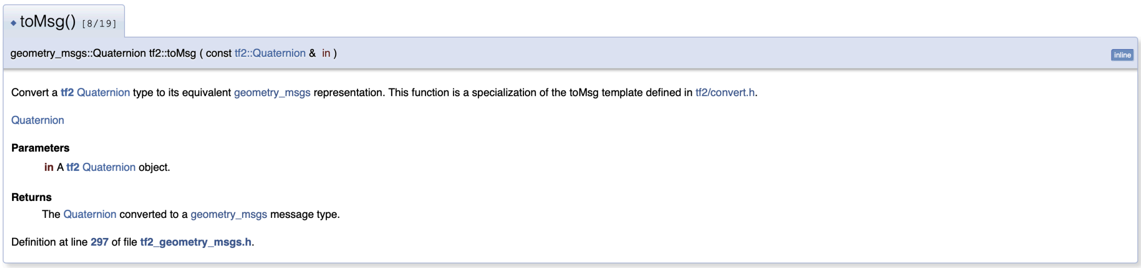
     Definition at line 313 of file tf2_geometry_msgs.h.

     Figure 1: tf2 Quaternion doc

   - **Assume we have just estimated our robot's newest rotation and it's saved in a variable called `quat_tf` of type `tf2::Quaternion`. Write one line of C++ code to convert it to a `geometry_msgs::Quaternion` type. Use `quat_msg` as the name of the new variable.**

```
geometry_msgs::Quaternion quat_msg = tf2::toMsg(quat_tf);
```

More specifically, we can find the official documentation of `toMsg()` in the same link as `fromMsg()`:


◆ toMsg() [8/19]

geometry_msgs::Quaternion tf2::toMsg (const tf2::Quaternion & in) inline

Convert a tf2 Quaternion type to its equivalent geometry_msgs representation. This function is a specialization of the toMsg template defined in tf2/convert.h.

Quaternion

Parameters
in A tf2 Quaternion object.

Returns
The Quaternion converted to a geometry_msgs message type.

Definition at line 297 of file tf2_geometry_msgs.h.

Figure 2: geometry_msgs Quaternion doc

- **If you just want to know the scalar value of a `tf2::Quaternion`, what member function will you use?**

```
double scalar = quat_tf.getW();
```

More specifically, we find the official documentation of `getW()` here:


◆ getW()

const TF2SIMD_FORCE_INLINE tf2Scalar& tf2::Quaternion::getW ( ) const inline

Definition at line 348 of file Quaternion.h.

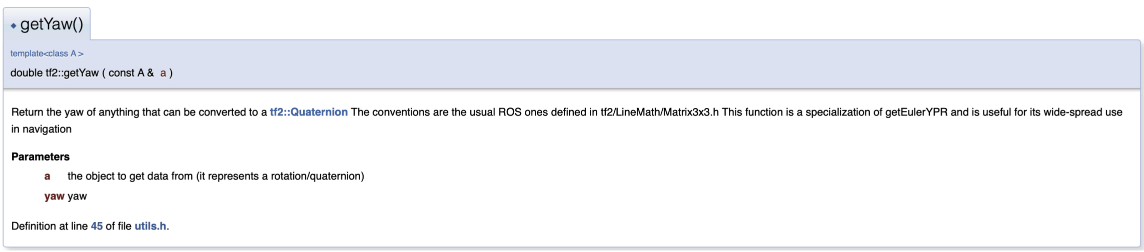
Figure 3: Quaternion get_w doc

2. **CONVERSION**

- **Assume you have a `tf2::Quaternion quat_t`. How to extract the yaw component of the rotation with just one function call?**

```
double yaw = tf2::getYaw(quat_t);
```

More specifically, the doc of `getYaw()` is shown at this page:


◆ getYaw()

template<class A >
double tf2::getYaw (const A & a)

Return the yaw of anything that can be converted to a tf2::Quaternion The conventions are the usual ROS ones defined in tf2/LineMath/Matrix3x3.h This function is a specialization of getEulerYPR and is useful for its wide-spread use in navigation

Parameters
a the object to get data from (it represents a rotation/quaternion)
yaw yaw

Definition at line 45 of file utils.h.

Figure 4: Quaternion get_yaw doc

- **Assume you have a `geometry_msgs::Quaternion quat_msg`. How to you convert it to an Eigen 3-by-3 matrix? Refer to this for possible functions. You probably need two function calls for this.**

```
#include <tf2_eigen/tf2_eigen.h>

Eigen::Quaterniond eigen_quat;

// The first function to call
tf2::fromMsg(quat_msg, eigen_quat);

// The second function to call
Eigen::Matrix3d eigen_mat3 = eigen_quat.toRotationMatrix();
```

More specifically, the doc of `toRotationMatrix()` can be found here:

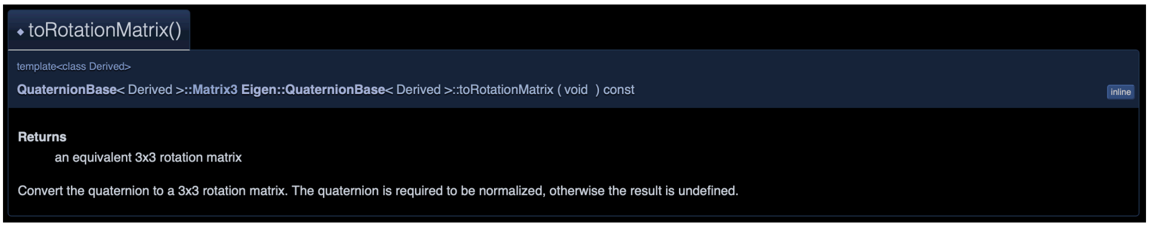
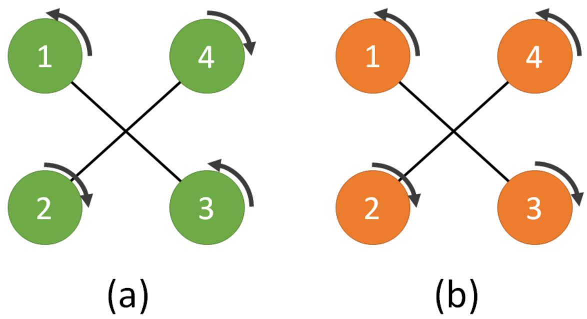


Figure 5: Eigen toRotationMatrix doc

2.1.2 Modelling and control of UAVs

1. **STRUCTURE OF QUADROTORS**



The figure above depicts two quadrotors (a) and (b). Quadrotor (a) is a fully functional UAV, while for Quadrotor (b) someone changed propellers 3 and 4 and reversed their respective rotation directions.

Show mathematically that quadrotor (b) is not able to track a trajectory defined in position $[x, y, z]$ and yaw orientation Ψ.

2. **CONTROL OF QUADROTORS**
Assume that empirical data suggest you can approximate the drag force (in the body frame) of a quadrotor body as:

$$F^b = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.2 \end{bmatrix} \left(v^b\right)^2$$

With $\left(v^b\right)^2 = \left[-v_x^b|v_x^b, -v_y^b|v_y^b, -v_z^b|v_z^b\right]^T$, and v_x, v_y, v_z being the quadrotor velocities along the axes of the body frame.

With the controller discussed in class (see referenced paper [1]), describe how you could use the information above to improve the tracking performance.

2.2 Team Work

2.2.1 Trajectory tracking for UAVs

2.2.2 Launching the TESSE simulator with ROS bridge

2.2.3 Implement the controller

2.2.4 Simulator conventions

2.2.5 Geometric controller for the UAV

3 Reflection and Analysis

4 Conclusion

5 Source Code