Managing Coordinate Frame Transformations in a Distributed, Component-Based Robot System

University of Applied Sciences Ulm
Department of Computer Science
Master Course Information Systems

Master Thesis
by
Manuel Wopfner

November 22th, 2011

Advisor: Prof. Dr. Christian Schlegel
Co-Advisor: Prof. Dr. Klaus Baer
Declaration of Originality

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Ulm, November 22th, 2011

Manuel Wopfner
Abstract

Autonomous mobile robots in general and service robots in particular consist of many actuators and sensors, which allow the robot to perceive and to interact with the world. Each of these actuators and sensors define an own coordinate frame. An actuator can change the transformation between two frames, whereas the data captured by a sensor is always relative to the sensor frame. If now, applications like collision avoidance or manipulation of objects should be done, sensor data has to be transformed from the sensor frame to another frame. To do this, a transformation tree, which describes the relation of frames to each other is needed. The tree allows to determine the single transformations which must be applied to get from one frame to another.

In a component-based robot system the information of the current transformations between the frames has to be transferred to all interested components. Therefore the robotic middleware (resp. framework), has to provide methods to do this. Further the quality of a transformation is not always the same. For example, it depends on the position uncertainty of the actuator and on the frequency at which the current position is captured.

The focus of this work is laid on the quality measurement for transformation chains and on the description of the data flow in component-based robotic systems. This description allows to understand the system, to compare it with other systems and to find bottlenecks in it. Further the quality measurement allows the system integrator, who builds a robotic system out of components, to see if his desired configuration gives the quality needed for a specific application. The quality is defined as the position uncertainty of a frame in 3D space modeled as a multidimensional Gaussian normal distribution. This thesis also presents methods to increase the quality of a transformation by adjusting parameters like the frequency at which the position of an actuator is captured.

Examples and a simulation of a system verify that the proposed methods are usable and general enough to describe all elements of the system relevant for managing transformations in a component-based robotic system.
Acknowledgements

I like to thank everyone who directly or indirectly contributed to this work.

First of all, I gratefully thank my thesis adviser Prof. Dr. Christian Schlegel for the opportunity to work in the ZAFH Servicerobotik the last years. I found it to be an interesting research area and I have learned a lot during this time. He gave me the chance to do this thesis on coordinate frame transformations in robotics and helped me to keep the focus in my research.

I would also like to thank Prof. Dr. Klaus Baer, who spontaneously agreed to act as co-adviser for my thesis.

Furthermore, special thanks go to my colleagues at the ZAFH Servicerobotik at the University of Applied Sciences Ulm for many fruitful discussions and the encouragement to do this work on transformations. In particular, I want to thank Siegfried Hochdorfer who helped me with some math and always had an open ear for questions.

Further, I want to thank my family and friends for supporting me during this work. Special thanks go to my long-suffering wife Claudia for her patience in the last year, when I spent nights and weekends at the study desk. Especially in the last weeks she relieved me of many of my everyday duties. Finally, eternal thanks to my God and Savior, Jesus Christ.
Contents

1 Introduction 1
  1.1 Introduction and Motivation ............................................. 1
  1.2 Overview of the Transformation Problem Space ........................... 2
  1.3 Thesis Outline and Contribution ......................................... 4

2 Related Work 5
  2.1 Coordinate Frame Transformations ....................................... 5
  2.2 Component-Based Robotic Middlewares .................................... 6
    2.2.1 Robot Operating System (ROS) ...................................... 6
    2.2.2 OROCOS .................................................................... 7
    2.2.3 Robot Technology Middleware (RT-Middleware) ...................... 7
    2.2.4 SMARTSOFT ............................................................. 8

3 Fundamentals 9
  3.1 Different Types of Joints .................................................. 9
  3.2 Transformations .............................................................. 10
    3.2.1 Matrices - Homogeneous Coordinates .................................. 12
    3.2.2 Euler Angles ................................................................ 12
    3.2.3 Quaternions .................................................................. 13
    3.2.4 Comparison of Methods for Rotation ................................. 14
  3.3 Time Synchronization .......................................................... 15
    3.3.1 NTP (Network Time Protocol) ........................................ 15
    3.3.2 PTP (Precision Time Protocol) ....................................... 16

4 Method 19
  4.1 Use Cases ................................................................. 19
    4.1.1 Collision Avoidance in 2D ............................................ 19
    4.1.2 Collision Avoidance in 3D ............................................ 20
    4.1.3 Using arm mounted sensors .......................................... 22
  4.2 Analysis of Requirements .................................................. 23
  4.3 Abstract System Description ............................................. 25
  4.4 Quality of Transformations ............................................... 28
    4.4.1 Modelling of Joints ..................................................... 29
    4.4.2 Mapping Joint Models to Cartesian Coordinates .................. 29
    4.4.3 Quality of Transformation Chains ................................... 32
    4.4.4 Position Prediction vs. Publishing Frequency ..................... 34
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5 Client/Server Separation</td>
<td>35</td>
</tr>
<tr>
<td>4.5.1 Validity of TF Data</td>
<td>36</td>
</tr>
<tr>
<td>4.5.2 Specification of the Communication Object</td>
<td>38</td>
</tr>
<tr>
<td>4.6 Constructing TF Systems</td>
<td>39</td>
</tr>
<tr>
<td>4.6.1 System Modelling</td>
<td>39</td>
</tr>
<tr>
<td>4.6.2 Calculating the Quality of a Joint</td>
<td>41</td>
</tr>
<tr>
<td>4.6.3 Calculating the Quality of a Transformation Chain</td>
<td>43</td>
</tr>
<tr>
<td>5 Results</td>
<td>49</td>
</tr>
<tr>
<td>5.1 Simulation</td>
<td>50</td>
</tr>
<tr>
<td>5.2 Discussion of the Results</td>
<td>52</td>
</tr>
<tr>
<td>6 Summary and Future Work</td>
<td>55</td>
</tr>
<tr>
<td>6.1 Summary</td>
<td>55</td>
</tr>
<tr>
<td>6.2 Future Work</td>
<td>55</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Introduction and Motivation

Today, service robotic systems consist of an increasing number of sensors and actuators. Thus for example, the state-of-the-art robot Asimo has 34 DOFs and the PR2 has about 24 DOFs depending on the equipment (see Figure 1.1). Further these robots have lots of different sensors like laser rangers, stereo cameras, accelerometers, microphones, pressure sensors and so on. Each of these actuators and sensors define an own coordinate frame. Thus it gets obvious that a modern robot is a complex system where many different coordinate frames have to be handled.

The difficulty is not the number of different coordinate frames, but that data has to be transformed between these coordinate frames. One well known example where such a transformation is needed is collision avoidance. There a laser scan, captured in the laser frame, has to be transformed into the base frame, so that the algorithm knows where the obstacles are relative to the robot base. There are many other examples where coordinate transformations are needed. In general it can be said, that in almost any case where sensor data is processed a transformation is needed.

![Figure 1.1: Three state-of-the-art robots.](image-url)

(a) Asimo from Honda. [12]  (b) PR2 from Willow Garage. [9]  (c) Care-O-Bot from Fraunhofer IPA. [15]
Today where the complexity of systems is continuously increasing it is no longer possible to run everything in one process. Large systems, like the ones in robotics, are often developed by different people or institutes. Thus the integration of other software parts into the own monolithic system is difficult and can cause inescrutable errors which are hard to debug. Further if one part of the system fails the whole system can fail. Thus it is important to divide the system into smaller parts, which do not effect each other.

In software engineering, and in the last years also in robotics, a solution for handling complex systems is component-based software engineering. There, software is split up into different components where each component defines an interface and other components only rely on this interface. This allows to reuse the components in different systems because there are no fixed dependencies between them. A simple system consisting of three components, which enable collision avoidance is shown in Figure 1.2. More details on component-based engineering in robotics are given in [4] and [5].

![Figure 1.2: Components which allow to do collision avoidance.](image)

Component-based software engineering is a good method to handle the complexity of a system, because it assigns responsibilities to components. In the case of coordinate frame transformations it allows to specify which components provide which coordinate frames and which components need this information. Nevertheless, the main challenge that remains is how to get the information to the right components and how good the quality of a transformation chain is.

### 1.2 Overview of the Transformation Problem Space

This section gives an overview of the transformation problem. It shows how transformations can be represented in an abstract way and clarifies some terms related to transformations which are used in this work.

A robot can be described as a set of links and joints, where links are connected through joints. An example for such a representation is shown in Figure 1.3. The figure also shows coordinate frames whereas the transformation between the frames is modified by the joints.

In a robotic system, where each coordinate frame has exactly one or no parent, the frames build a tree structure called transformation tree or in short tf tree. Such a tree for a simple robot is shown in Figure 1.4. The tree shows a robot base, equipped with a laser, a manipulator and a pan tilt unit. Another laser is mounted on the manipulator. On the ptu there is a Time of Flight camera and a normal color camera. The root element in the tree is the world_frame which specifies a reference point in the world. Linked with this frame is the base_frame which is the root frame on the robot. In this thesis only frames on the robot, which are all below the base_frame in the tf tree are considered. Nevertheless it is also possible to apply the methods shown in this thesis to all other frames.
1.2. OVERVIEW OF THE TRANSFORMATION PROBLEM SPACE

If collision avoidance should be done by using the laser scanner on the arm, the laser scan captured in the \textit{laser\_frame} must be transformed into the \textit{base\_frame}. This includes all single transformations in the branch from the base to the laser and is called a transformation chain (\textit{tf chain}). A \textit{tf chain} always consists of all transformations which have to be applied to get from one frame to another frame.

![Abstract robot model consisting of links and joints.](image1)

**Figure 1.3:** Abstract robot model consisting of links and joints.

The links between two frames can be divided into two categories. There are \textit{static frames} and \textit{dynamic frames}. Synonymously it could also be said, that the transformations are \textit{static} or \textit{dynamic}. \textit{Static frames}, like the name says, are always fixed and do not change over time. The quality of the transformation is of course also static and does not change. In contrast thereto, \textit{dynamic frames} have

![General tf tree.](image2)

**Figure 1.4:** General tf tree.
two states. They can be fixed or they are moving. This is an important distinction, because it has effects on the quality of a transformation. If the frame is in fixed state the quality of the transformation is only effected by the position accuracy of the actuator which moves the frame. In contrast thereto, if the frame is in moving state the quality is for example further effected by the uncertainty in velocity and acceleration.

1.3 Thesis Outline and Contribution

The major contribution of this thesis is a method to describe systems on an abstract level and to enable developers to tune the system to get the needed quality for the transformation. The focus of this thesis is laid on the definition of elements which are necessary to describe component-bases systems and the data flow in the system related to transformations. This is a major step to discuss the design of new and existing systems and allow to compare them to each other. Further a quality measurement for transformation chains based on position uncertainties in 3D Cartesian space is introduced. This measurement allows to find the transformations which add the most uncertainty in the chain and enables the developers and system integrators to tune the system for their needs. Finally the values which describe a transformation and thus must be exchanged by the components to perform transformations between frames are presented.

This introduction has given an overview on the transformation problem space in robotics and the topics that are addressed by this work. The rest of the thesis is organized as follow:

Chapter 2: Represents general work related to coordinate frame transformation. Further component-based frameworks like ROS where coordinate frame transformation is addressed are presented.

Chapter 3: Explains the fundamentals needed to understand the methods explained in Chapter 4. This mainly includes the introduction to transformations and their different representations.

Chapter 4: This chapter presents use-cases which show prominent examples where transformations are needed. Then requirements for a component-based robotic system which evolve from the use-cases are presented. Ongoing methods to satisfy these requirements are described.

Chapter 5: Shows the modelling and simulation of an example system with the introduced methods, to validate their applicability for managing coordinate frames.

Chapter 6: Finally a conclusion is drawn and an outlook for future work is given.
Chapter 2

Related Work

This chapter presents work related to this thesis. At first, general applications where coordinate frame transformations are used will be presented. Afterwards component-based robotic middlewares which partly support transformations are discussed. These middlewares and how coordinate frame transformations are used in different applications are of interest for this work, because they show what already exists in this area and which topics still must be addressed.

2.1 Coordinate Frame Transformations

Coordinate frames and the transformation of data between different coordinate frames are widely used in many applications. Coordinate frame transformation provide the mathematic foundation to solve the problems which occur in these applications. Being a basic mathematic tool coordinate systems and their transformation are described in many math books like [17][20]. In the following, examples of areas where transformations are used are given. Further the way how coordinate frame transformations are managed in these applications is judged in terms of the use in robotic systems.

An application where transformations are used for a long time is celestial mechanics [14]. There they are used to transform observational quantities from one coordinate frame to another. These could for example be two observatories on the earth exchanging data of some observation. Celestial mechanics are a difficult and complex topic, that also deals with uncertain observations. In most cases not the Cartesian coordinate systems is used, but Geodetic or Geocentric coordinate systems are used. In general some methods like the use of uncertainties can also be applied in robotics, but celestial mechanics provide far more methods like the determination of orbits which are not needed in robotics.

Another application where coordinate systems are widely used is aerospace [19]. There the position of an airplane is normally given relative to the coordinate system of the earth. However if a spacecraft is sent into deep space a heliocentric system is preferred. Further there are additional coordinate systems like Body Coordinates or Wind Coordinates and there is also an coordinate system for navigation. Altogether aerospace is a complex application where coordinate systems are used. Aerospace has to deal with other problems like wind which does not occur in service robotics. Thus the way how coordinate systems are used in aerospace can not be directly applied in robotics.

Further CAD systems like AutoCAD [2] or 3D modelling suits like Blender [3] use coordinate systems. There the position of bodies in relation to each other are described. The transformations have precise values which means that there is no uncertainty included. In contrast thereto robotic systems include sensors and actuators which introduce uncertainty. Further in robotics the transformation is
not known at every point in time, because measurements can not be taken at an infinite short time period.

Simulators like Gazebo [23], define a coordinate system for each object they have to simulate. During the simulation they apply physical rules and they use additional control inputs to calculate the position of objects for the next time step. The calculated positions are absolute, but the simulator can provide uncertain measurements for the user. This uncertainty which represents measurement noise is necessary to simulate real devices like a laser ranger, which in reality also provide noisy measurements. Simulators like Gazebo use transformations in a very similar way like it is done in robotic systems. However in Gazebo everything runs in one process, thus it has not to deal with the problem of concurrent and distributed systems. Further, since it is a simulation and therefore a discrete-time system, all transformations at each time step are known. In contrast thereto robotic systems are sampled-data systems with continuous time where the transformation is not known for every point in time.

A further application where transformations are used are robotic arms like the ones from KUKA [18] which are used in industrial robotics. These arms consist of several joints connected with each other. Each of these joints define a own coordinate frame. All these frames together define a tf chain. With the values of each coordinate frame the position of the end effector also known as tool center point (tcp) can be determined. The tcp is only described by his translation and rotation in relation to the root frame of the arm. There is no additional value that describes the uncertainty of the tcp position at a specific point in time. The vendors only give a general repetitive accuracy. Further the transformations of all joints are known at the same point in time. In contrast thereto the transformations in a component-based and distributed system are only known at different points in time and thus has to be merged in another way.

2.2 Component-Based Robotic Middlewares

Today there exist many different component-based frameworks for robotic systems. In the last year the most prominent representative is ROS which is mainly developed by Willow Garage [10]. These different frameworks support coordinate frame transformations at different levels. Some of them have special libraries or other build in support. Other frameworks only provide message exchange between different components where any kind of data can be exchanged with no special support for transformations. In these frameworks the developer has to handle the transformations by himself.

2.2.1 Robot Operating System (ROS)

The Robot Operating System (ROS) [24] has the biggest build in support for transformations. There a package exists called tf and a new experimental redesigned package called tf2. The package allows to keep track of multiple coordinate frames over time and can transform points etc. between these frames at a specific point in time. Further the package manages the distribution of the frame transformations in the whole system and saves the history in each component that is interested in the transformation. In ROS the messages for tf are separated from the other messages. This means, that a tf message is never included in another message like a laser scan message. If a point etc. should be transformed, only the name of the source and target frame as well as the point in time have to be known. The tf package provides functions for transforming most data types and if the transformation can not be performed an error message is returned.

Further the package can read Unified Robot Description Format (URDF)[32] files which allows
easy definition of the robots kinematic. To publish the frames a robot state publisher is provided which determines the tf frames from the raw joint data.

The provided tools in ROS greatly support the developer with the use of coordinate frames, because the framework takes care of how the transformations are done and how the information is exchanged between components. The developer just has to specify the transformation his interested in. However, ROS handles all transformations in the same way and does not divide between static and dynamic frames. There is also no way in ROS to determine the quality of a transformation chain to find out if it satisfies the requirements of the application. Further it always publishes all TF Data to all components using tf, also if the TF Data is not needed to perform the transformation the component is interested in.

The ROS framework inspired this work through its powerful build in support for transformations and the use of URDF to describe the kinematics of a robot. Further the disadvantages of ROS showed topics where improvements are needed.

### 2.2.2 OROCOS

The Open Robot Control Software (OROCOS) project [6] is a framework which allows components to run in hard real time and on multiple computers. It comes with a special Kinematics and Dynamics library called KDL. This library supports the modelling and computation of kinematic chains out of primitives like segments and joints. After the construction of the model, transformations between frames can be easily calculated by different solvers. Right now only forward kinematic solvers are implemented but own inverse kinematic solvers can be added. Further the use of the KDL library is not only limited to robotics, it is also useful for computer animated figures or human models.

The KDL library in OROCOS is a good tool to solve kinematic chains but OROCOS has no special build in support to distribute information of the tf tree between different components. Thus the developer has to decide on his own how this information should be exchanged and how components deal with this information. Further like in ROS there is also no method the determine the quality of a transformation chain.

### 2.2.3 Robot Technology Middleware (RT-Middleware)

OpenRTM-aist [1] is an open source implementation of an RT-Middleware and fully compatible with the Robot Technology Component Specification from OMG [25]. The framework provides a component model for distributed RT systems and also defines standard interfaces for the components. It supports components at three different levels. Components for fine granularity level, such as a motor or sensor. Components of middle-fine granularity level such as a mobile robot with some sensors and rough granularity level such as a humanoid robot.

Further each component has an activity which can be in different states like INITIALIZE, READY, ACTIVE, ERROR. The states allows entry, do and exit actions which make it possible to control the action of many components similarly. In the paper where RT-Middleware [1] was proposed it is said that coordinate system handling is indispensable for RT-Middleware and it should be either realized as a service or imported into the RTM specification. Until now its not available in the official RTM release neither as a service nor as an import into the specification.

Nevertheless there exist implementations in form of components in projects where RTM is used. One example is [26] where an intelligent space called iSpace is developed on the base of OpenRTM-aist. For the iSpace several utility components were implemented where one is responsible for coordinate frame transformation. The input for this component is the data that should be transformed and
the transformation parameters. As a result the transformed data is returned.

The implemented utility component for the iSpace is tailored for the use in the iSpace and thus can not be reused in other applications. This proprietary solution shows the need for a general solution by the framework.

2.2.4 SmartSoft

SmartSoft was first described in [27] and later in more detail in [28]. It is a component-based framework with seven Communication Patterns that enable easy communication between components. Each pattern defines the communication mode, provides predefined access methods and hides all communication and synchronization issues. Communication Objects define the data which is exchanged between components and the combination of a Communication Pattern and a Communication Object build a Service that the component offers. Further each component defines a state automaton which represents the life-cycle of the component. It also allows to switch the component from external into a different state, for example activate/deactivate a camera.

SmartSoft does not come with a special library for coordinate transformations, but the available Communication Objects on SourceForge\footnote{http://smart-robotics.sourceforge.net} have special access methods which allow to transform the data to basic coordinate frames like base frame or world frame. Thus a basic support is given but if special transformations are needed the developer has to do them by himself.

The way transformations are supported in SmartSoft makes it really easy for the developer to use them if he is only interested in basic transformations. However the embedding of TF Data into other messages like a laser scan message introduces problems which does not occur if the data is separated. For example the implementation of Communication Objects gets more difficult, because all the logic has to be added to the Communication Object. Further also the components get more complex. For example a component for a laser ranger, which normally just publishes the laser data, now also have to put the TF Data into the laser scan message. This TF Data must be acquired from another component. Thus additional ports and connections to other components have to be added which increases the overall system complexity.

SmartSoft has inspired this work by the clear separation of components by Communication Objects. Further together with ROS it showed the need of an abstract method to describe different systems, to allow a comparison of them.
Chapter 3

Fundamentals

In this chapter fundamentals, which help to understand the content of the further chapters, are presented. After looking at the different types of joints, the term Transformation is introduced which is a key element of the his work. Later the problem of Time Synchronization and how clocks can be synchronized are described. This is interesting, because time plays an important role in concurrent and distributed systems.

3.1 Different Types of Joints

A robot is built out of joints and links whereas the links are connected through joints. To describe a robot the Unified Robot Description Format (URDF) [32] can be used. It allows to describe the robot with all its kinematics and dynamics in a XML format. A lot of robots like the PR2, the Care-O-bot or the Nao are modeled in URDF. For this thesis the different joint types which are supported by URDF are interesting, because they have proven to allow a description of arbitrary robot setups.

If the URDF description should be used for building the tf tree of a robot, a mapping of joint values to coordinate frames must be performed. Figure 3.1 shows two links which are connected by

![Figure 3.1: Relationship between frames and joints.](image-url)
one joint. In the figure two coordinate frames namely the **Parent frame** and the **Child frame** can be seen. Depending on the type of joint, the joint can rotate about or move in direction of one axis of the **Joint frame**. The **Joint frame** can also be called **Child frame**. To actually convert the joint value into a coordinate frame the *default translation* and *rotation* from the **Parent frame** to the origin of the **Child frame** has to be known. Further the additional translation or rotation introduced by the joint has to be added. Together these two transformations form the transformation from the **Parent frame** to the **Child frame** and can be used in the *tf tree*.

Until now only *hinge* and *prismatic joints* were considered, but in *URDF* even more joints are specified. It is possible to convert these other joints to coordinate frames in a similar way. The different types of joints supported by *URDF* are:

- **revolute joint**: This hinge joint rotates around an axis and has a limited range.
- **continuous joint**: This is a continuous hinge joint that rotates around an axis with no upper and lower limits.
- **prismatic joint**: This is a linear joint that moves along an axis. It has a limited range.
- **planar joint**: This joint allows motion in a plane.
- **fixed joint**: This is not really a joint because all degrees of freedom (translation and rotation) are locked. Thus the joint cannot move.
- **floating joint**: This is also not really a joint, because all degrees of freedom are free.

### 3.2 Transformations

There are different transformations on point-sets in $\mathbb{R}^3$. The ones which are interesting for robotic applications are the **Translation** and the **Rotation** Transformation. If, from now own, it is spoken about transformations these two are meant. The **Translation** simply defines a shift of an object within a viewing frame in accordance to the rule

$$ P = (x, y) \rightarrow P' = (x + h, y + k) $$

Figure 3.2 shows an example, where a square in $\mathbb{R}^2$ is shifted in x and y direction.
This translation can be written in matrix form [17, p.335] as

\[
P' = P + T
\]

where

\[
T = \begin{bmatrix} 2.5 \\ 1 \end{bmatrix}
\]

So

\[
P' = P + \begin{bmatrix} 2.5 \\ 1 \end{bmatrix}
\]

where

\[
P = \text{original point-set}
\]

\[
P' = \text{translated point-set}
\]

In contrast thereto a Rotation subjects an entire point-set to a rotation about the origin through an angle \( \theta \). This is done in accordance with the rule [17, p.336]

\[
P = (x, y) \rightarrow P' = (x', y')
\]

where

\[
x' = x \cos \theta - y \sin \theta
\]

\[
y' = x \sin \theta - y \cos \theta
\]

Figure 3.3 shows such a Rotation in \( R^2 \), where a square is rotated about the origin through \( \theta \).

There are even more than the transformations described above. For example there is a further Scale transformation, which allows scaling of point-sets. In the case of a square this would mean the scaling of the square in size. However these transformations only play a minor role in robotics, and especially in coordinate frame transformation. Thus they are not considered here.

Translations can simply be described as a vector or as a matrix if homogeneous coordinates are used. In contrast thereto Rotations are more complex to describe. There are three popular methods namely Matrices, Euler Angles and Quaternions to describe Rotations in \( R^3 \). In the next sections these methods will be introduced in short to give a basic understanding on how they work. At the end a comparison between the different methods is given, that shows which method is useful for which application.
3.2.1 Matrices - Homogeneous Coordinates

The representation of rotations as matrices is probably the best known one, because it is taught during university education. Furthermore, if homogeneous coordinates are used, it is easy to combine rotation and translation in a consistent way. The following equations just recap how these matrices look like:

\[ R_x^\phi = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & \sin\phi & 0 \\ 0 & -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

\[ R_y^\theta = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

\[ R_z^\psi = \begin{bmatrix} \cos\psi & \sin\psi & 0 & 0 \\ -\sin\psi & \cos\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

\[ T = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

The first three matrices describe rotations about the x-axis, the y-axis and the z-axis. The last matrix is a homogeneous matrix for an arbitrary translation. With these matrices and the rules for matrix calculation it is possible to build arbitrary transformation sequences which can be applied on points.

3.2.2 Euler Angles

The Euler Angles were introduced by the famous mathematician Leonard Euler (1707-1783). In his work in celestial mechanics he stated and proved the theorem that:

*Any two independent orthonormal coordinate frames can be related by a sequence of rotations (not more than three) about coordinate axes, where no two successive rotations may be about the same axis.* [17, p. 83]

This theorem states, that one frame can be rotated into another frame through rotations about successive coordinate axes. The angle of rotation about a coordinate axis is called an Euler Angle and a sequence of rotations is called a Euler Angle Sequence. The theorem of Euler which says that two successive rotations must be about different axes allows twelve Euler angle-axis sequences. These axis-sequences are:

- xyz
- xzy
- xzx
- yzx
- yxy
- zxy
- zyx
- zyz
- zxx
- zzy
- zyz

The sequences are read from left to right. For the sequence xyz this means a rotation about the x-axis, followed by a rotation about the new y-axis, followed by a rotation about the newer z-axis. In case of
3.2. TRANSFORMATIONS

the popular Aerospace Sequence, which is the zyx sequence, it can be expressed as a matrix product as

\[ R = R_z^\psi R_y^\theta R_x^\phi \]

\[ = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & \sin \phi \\ 0 & -\sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{bmatrix} \begin{bmatrix} \cos \psi & \sin \psi & 0 \\ -\sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \]

This equation states, that the first rotation is through the Heading angle \( \psi \) about the z-axis, followed by a rotation through the Elevation angle \( \theta \) about the new y-axis. The final rotation is through the Bank angle \( \phi \) about the newest x-axis.

Euler Angles are a simple way to represent rotation sequences. They allow to describe the rotation of an arbitrary frame into any other frame. Nevertheless they have two main disadvantages [8, p. 156]. First the representation for a given orientation is not unique and second, the interpolation between two angles is difficult. The phenomenon that comes from this disadvantages is known as Gimbal lock.

3.2.3 Quaternions

Quaternions were invented by Hamilton in 1843 as a so-called hyper-complex number of rank 4. They are another method to represent rotations, which solve the problems of the Euler Angles. The most important rule for quaternions is

\[ i^2 = j^2 = k^2 = i j k = -1 \]

A quaternion is defined as the sum

\[ q = q_0 + q = q_0 + iq_1 + jq_2 + kq_3 \]

where \( q_0 \) is called the scalar part, and \( q \) is called the vector part of the quaternion. The scalars \( q_0, q_1, q_2, q_3 \) are called the components of the quaternion. Further \( i, j \) and \( k \) are the standard orthonormal basis in \( R^3 \) and defined as

\[ i = (1, 0, 0) \]
\[ j = (0, 1, 0) \]
\[ k = (0, 0, 1) \]

The quaternion rotation operator \( L_q \) is defined as

\[ w = L_q(v) = qvq^* \]

where \( w, v \) are vectors of \( R^3 \), \( q \) is a quaternion and \( q^* \) is the complex conjugate of this quaternion. For the equation to be valid, the vector \( v \) has to be converted into a pure quaternion which is \( v = 0 + v \). The algebraic action of Equation 3.1 is illustrated in Figure 3.4.

In this section only a brief overview of quaternions with the basic definitions were given. A complete explanation of quaternions can be found in [17].
3.2.4 Comparison of Methods for Rotation

The methods described above are all used in robotics, because they fulfill different requirements and thereto are used for different applications. A good comparison is given in [8, p. 179] and the most important differences are listed in Table 3.1. This table shows, that each representation has its advantages and disadvantages, thus it is a good choice to always use the appropriate representation for the current application. Conversions between the different representations are of course possible and are for example described in [8, p. 180].

<table>
<thead>
<tr>
<th>Task/Property</th>
<th>Matrix</th>
<th>Euler Angles</th>
<th>Quaternion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rotating points between coordinate spaces (object and internal)</td>
<td>Possible</td>
<td>Impossible (must convert to matrix)</td>
<td>Impossible (must convert to matrix)</td>
</tr>
<tr>
<td>Concatenation or incremental rotation</td>
<td>Possible but usually slower than quaternion form</td>
<td>Impossible</td>
<td>Possible, and usually faster than matrix form</td>
</tr>
<tr>
<td>Interpolation</td>
<td>Basically impossible</td>
<td>Possible, but aliasing causes Gimbal lock and other problems</td>
<td>Provides smooth interpolation</td>
</tr>
<tr>
<td>Human interpretation</td>
<td>Difficult</td>
<td>Easy</td>
<td>Difficult</td>
</tr>
<tr>
<td>Storing in memory</td>
<td>Nine numbers</td>
<td>Three numbers</td>
<td>Four numbers</td>
</tr>
<tr>
<td>Representation is unique for a given orientation</td>
<td>Yes</td>
<td>No - an infinite number of Euler angle triples alias to the same orientation</td>
<td>Exactly two distinct representations for any orientation</td>
</tr>
<tr>
<td>Possible to become invalid</td>
<td>Can be invalid</td>
<td>Any three numbers form a valid orientation</td>
<td>Can be invalid</td>
</tr>
</tbody>
</table>

Table 3.1: Comparison of matrices, Euler angles, and quaternions. [8, p. 179]
3.3 Time Synchronization

State of the art service robots have normally more than just one PC on board. Thus the clock of the different PCs have to be synchronized if data from two or more PCs have to be handled according to time. For coordinate frame transformation this is the case where transformations can be published by different computers and have to be matched according to their time stamp.

The general problem with multiple computers as described in [31, p. 239] is, that the clock of each PC has a drift. The drift is produced by production tolerances of the oscillators and by environmental influences such as different temperatures. Thus clocks on different PCs will always drift apart.

If \( t \) is the reference time and \( C_p(t) \) is the value of the clock on machine \( p \), the equation \( C_p(t) = t \) would be true for all \( p \) and all \( t \) if there would be no drift. Further \( C'_p(t) = \frac{dC_p}{dt} \) ideally should be 1. The frequency of a clock \( p \) at time \( t \) is \( C'_p(t) \), the skew of a clock is defined as \( C'_p(t) - 1 \) and donates how much the frequency differs from the perfect clock. \( C_p(t) - t \) simply defines the offset between the clock \( p \) and the reference clock. The maximum drift rate \( \rho \) is specified by the manufacturer and the timer works in its specification if

\[
1 - \rho \leq \frac{dC_p}{dt} \leq 1 + \rho
\]

If two equal clocks are compared to the reference clock at time \( \Delta t \) after they were synchronized, they can not be more than \( 2\rho \cdot \Delta t \) time units apart. To guarantee that no two clocks ever differ more than \( \delta \), they must be synchronized at least every \( \frac{\delta}{2\rho} \) time units. The relationship of a slow, a perfect and a fast clock is shown in Figure 3.5.

![Figure 3.5: Relationship between clocks when they tick at different rates.](image)

3.3.1 NTP (Network Time Protocol)

The probably most used time synchronization protocol is the Network Time Protocol (NTP). It allows to synchronize clocks with a worldwide accuracy in the range of 1-50 milliseconds [31, p. 241]. To achieve this, a good estimate of the message delay has to be determined. Without the correction of the message delay, the synchronization could never get better than the delay itself. Due to the worldwide use, NTP divides servers into strata. A server with a high precision clock, such as an atomic clock is a stratum-1 server, whereas the clock itself is at stratum-0. A machine \( B \) only adjusts its clock if its stratum level is higher than the stratum level of the machine \( A \) it wants to synchronize. After synchronization the stratum level of \( B \) gets one higher than the level of \( A \). If for example the level of \( A \) is \( k \) the level of \( B \) will get \( k + 1 \). This structure allows efficient worldwide time synchronization.
Figure 3.6: Way to get the current time from a time server.

Figure 3.6 shows an example, where $A$ wants to synchronize to $B$. To do this, $A$ sends a request to $B$, stamped with the time value $T_1$. If the message is received, $B$ will record this time $T_2$. $B$ then sends a response to $A$ containing $T_2$ and $T_3$. Finally $A$ records the time $T_4$ when the message from $B$ was received. It is assumed that the message delay from $A$ to $B$ is roughly the same than from $B$ to $A$. This means that $T_2 - T_1 \approx T_4 - T_3$. Thus, in this example $A$ can estimate delay $\delta$ and its offset $\theta$ to $B$ as

$$\delta = \frac{(T_2 - T_1) + (T_4 - T_3)}{2}$$

and

$$\theta = T_3 + \delta - T_4.$$  \hspace{1cm} (3.2)

3.3.2 PTP (Precision Time Protocol)

To get better clock synchronization than achievable by NTP the Precision Time Protocol (PTP) was designed [13]. It is for example used in industry for industrial automation or robotic control. With special network cards and switches a synchronization accuracy of 10-100 nanoseconds is achievable. With normal network components and a software implementation of PTP a synchronization accuracy of 10-100 microseconds is possible [22].

The Precise Time Protocol synchronizes clocks between a Master and a Slave, similar to the Network Time Protocol. The Master provides the time and one or more Slaves synchronize to the Master. Therefore the Master sends Sync messages, typically every two seconds to the Slaves. Further the Slaves send less frequently, about one request per minute, Delay Request messages to the Master to determine the network delay, which is needed for time correction. The four timestamps needed for accurate time synchronization are $T_1$, $T_2$, $T_3$ and $T_4$ (see Figure 3.7) [22].

When the Master is sending the Sync message, $T_1$ is captured. $T_1$ is then later send to the Slave via the Sync Follow up message. The Slave captures $T_2$ when the Sync message is received and $T_3$ when it sends the Delay request message. When the Master receives the Delay request message, it records $T_4$ and sends it back to the Slave with the Delay Response message. The Slave can now calculate the delay $\delta$ with Equation 3.2 and the offset $\theta$ with

$$\theta = T_2 - \delta - T_1.$$  \hspace{1cm} (3.3)

The Precision Time Protocol allows sufficient time synchronization for applications on mobile robots where the normal frequency of data capturing is not higher than 50 Hz which is 20 ms. Thus the uncertainty introduced by the measurement error is far higher than the one introduced by the inaccuracy of the time synchronization. Further it is easy to setup PTP on the Linux operating system,
which is widely used in robotics. For Linux a daemon called PTPd exists, which works with normal network hardware. It is shown in [7] that a synchronization accuracy of 10-100 µs can be reached with PTPd.
Chapter 4

Method

At the beginning of this chapter some real world use-cases are presented. From the analysis of these use-cases evolving requirements are formulated. Afterwards elements which allow the abstract description of a component-based system are presented. Further a method to determine the quality of a transformation chain is shown. Finally some examples using the different elements and the quality measurement are presented, to show the generality of the proposed methods.

4.1 Use Cases

This section describes several use-cases where it is necessary to transform data between different coordinate frames. The selected use-cases have relevance for service robotics where a great number of actuators and sensors exist and where sensors are mounted on actuators. The chosen use-cases increase in complexity, starting with a setup where no actuators are included. The next use-cases mainly increase in the number of actuators involved in the transformation chain. Today humanoid robots have the probably most complex `tf tree`. However, the challenges are the same as for the third use-case, where a sensor mounted on an arm is considered. Thus these use-cases are a representative subset for the use of transformations in a robotic system.

Each of the use-cases contains a description of the use-case and a summary of the most important challenges occurring in this use-case. Based on these challenges the requirements for this work are derived in Section 4.2.

4.1.1 Collision Avoidance in 2D

One task every robot has to fulfill is collision avoidance. Therefore most robots are equipped with a planar laser scanner, like shown in Figure 4.2(a). This laser ranger is mounted at a fix position on the robot. In a component-based system, there are typically three components involved (see Figure 4.1). The base component is responsible for controlling the robot base, the laser component which publishes the newest laser scan and a collision avoidance component subscribing for laser scans. The latter component calculates the next collision free steering command for the base component based on the given sensor data.

To calculate the next steering command the current laser scan, which of course is captured in the laser_frame, has to be transformed into the base_frame. This transformation is necessary because the steering commands are always relative to the base_frame. As shown in Figure 4.2(b) the `tf tree` for this case is a very simple one. The tree has only two nodes, namely the base_frame
CHAPTER 4. METHOD

Figure 4.1: Components needed for collision avoidance.

and the laser_frame. These two frames are linked statically, because the laser is directly mounted on the base, with no actuator inbetween.

Figure 4.2: Hardware setup and tf tree of a statically mounted laser scanner.

Main challenge in this use-case:

- **Publish transformation once**: It is enough to publish the transformation only once to every interested component, because the transformation does not change over time. It is important that also components which are started later also receive the transformation once.

4.1.2 Collision Avoidance in 3D

The use-case described above only considers objects in the plane of the laser scanner. It is obvious that in this case not all collisions are avoided, thus collision avoidance should be performed in 3D. One way to do this is to mount a laser scanner on a tilt unit. Such a setup is for example used on the PR2 robot (see Figure 4.3(a)). In this setup the laser scanner is tilted continuously up and down and thereby creates a 3D point cloud of the environment. In this point cloud the drivable area is determined. In the component-based system above only the collision avoidance component has to be changed, the two other components can stay the same.
4.1. USE CASES

(a) Tilting planar laser scanner. [9]

(b) Tf tree for a tilting laser scanner.

Figure 4.3: Hardware setup and tf tree of a tilting laser scanner.

As it can be seen in Figure 4.3(b) the laser ranger is no longer mounted at a fix position on the robot. It now changes its position due to the movement of the tilt unit. This use-case is even more complex, because the captured laser scan no longer lies in the xy plane of the laser frame. It now lies in a plane rotated about the x-axis. Figure 4.4 illustrates this two cases.

Figure 4.4: Tilting laser scanner moves the laser scan from the xy-plane to a oblique plane.

Main challenges in this use-case:

• **Publish transformation continuously:** The transformation has to be published continuously to all interested components.

• **Low latency:** The transformation information should be delivered quickly to the interested components, so that they can do their calculations.

• **High synchronicity:** In the case of the tilting laser scanner, the moment when the laser scan and the pose of the tilt unit are captured, should be nearly the same. The amount of time between the point when the laser scan and the point when the tilt unit pose is captured influences the quality of the transformation.
4.1.3 Using arm mounted sensors

Different robots have cameras, lasers or other sensors mounted on their arm like shown in Figure 4.5(a)[33][21]. This allows them to perceive their environment in a more flexible way. If for example an object is standing on a table and the robot wants to look at the object from the side (to e.g. see the barcode) the robot can use its camera mounted on the arm. Another scenario, where a laser ranger can be used, is to capture a 3D point cloud from a shelf where the robot cannot see into with the sensor mounted on its head.

The *tf tree* for an arm mounted sensor can look like shown in Figure 4.5(b). The components which are involved to capture a point cloud (e.g. of a shelf) are shown in Figure 4.6. The *laser component* provides the laser scans captured by the arm mounted laser ranger. The *arm component* commands the manipulator and the *3d point cloud generator* creates the point cloud by orchestrating the two other components. This component is also responsible to provide the created point cloud to other components.

![Diagram of hardware setup and tf tree of a sensor mounted on an arm](image)

(a) Arm mounted camera. [9] (b) *Tf tree* for sensor mounted on an arm.

Figure 4.5: Hardware setup and *tf tree* of a sensor mounted on an robot arm.

![Diagram of components needed to capture a 3D point cloud](image)

Figure 4.6: Components needed to capture a 3D point cloud.
4.2. ANALYSIS OF REQUIREMENTS

To capture a 3D point cloud the arm has to pan the laser ranger over the shelf. This can be done in two different ways. The first one is to move the arm to a specified position, stop there, take the laser scan and then move on to the next position. This is the simpler approach since the whole transformation chain is fixed at the point in time the laser scan is taken. Another approach is to continuously take laser scans while the arm is moving. This is of course more complex, because the transformation chain changes over time. Further the same effect as described in Section 4.1.2 appears which makes it even more complex to accurately capture the whole 3D point cloud.

Main challenges in this use-case:

- Synchronize long transformation chains: Long chains of more than just one or two joints have to be synchronized. This is especially challenging if the TF Data is generated by components running at different rates.

- Consider fixed and moving state: The two different states at which a laser scan can be taken have to be considered, because this influences the accuracy of the pose where the scan was taken.

4.2 Analysis of Requirements

After the analysis of the use-cases described above different requirements evolve for a tf system. Further the use of a component-based system introduces some requirements which are not directly related to the transformation problem. Nevertheless only the use of a component-based system makes it possible to handle the overall system complexity of an advanced robotic systems.

I. Publish Static Transformation Once

Static Transformations have to be published only once to all interested components.

The values for a Static Transformation must only be sent once to all interested components, because they do not change over time. The interested components are the ones, where this transformation is a node in the tf chain, the component is interested in. However it is also possible to publish the transformation periodically, but with the drawback of wasted resources. If the TF Data is exchanged via publish/subscribe, the framework must ensure, that components which later subscribe to the message do get it.

II. Publish Dynamic Transformation Continuously

Dynamic Transformations have to be published continuously when they are in moving state.

If a Dynamic Transformation is changing, because the actuator is moving, the transformation has to be published with a previously defined frequency. This rate has to be chosen in a way, that the needed quality is provided. A higher frequency increases the quality of the transformation as it will be shown in Section 4.4.4.

III. Consider Fixed and Moving State

Fixed and Moving State of a joint have to be distinguished to achieve optimal resource usage.
For a *Dynamic Transformation* the two different states it can have must be considered. In the *moving state* the transformation has to be published continuously. If it is in the *fixed state* the transformation must not be published, until it goes to the *moving state*. It is also possible to handle both states in the same way by always publishing the data with the same frequency. However this can cause a lot of resource wastage, because many transformations are normally in the *fixed state*.

IV. Low Latency

The *Latency* between different components and for the transformation calculation must be low.

To allow the construction of high reactive systems, the latency for inter component communication as well as the time needed to calculate the transformation itself must be small. This requirement is mainly a requirement for the middleware and the implementation of the *tf library*. Thus this requirement is not considered in this thesis.

V. High Synchronicity

The *sensor data* and the *TF Data* which is needed to transform the sensor data should optimally be captured at the same point in time.

If the *tf frame* is in *moving state* its *TF Data* should be captured at the same time as the sensor data. Every time difference increases the position uncertainty of the sensor at the time when the sensor data was captured. This increase in uncertainty and the relation to time is explained in Section 4.4.1. In the *fixed state* the time difference has no influence on the position uncertainty, because there is no movement of the *tf frame*.

This requirement is one of the hardest to achieve, because each component runs in its own process and thus can not easily be synchronized. Further it is not enough to only synchronize the components in fact the different hardware actuators and sensors have to be synchronized. In reality the most appropriate way is to set the frame rate for the publishing of the *TF Data* to a rate which is sufficient to achieve the desired transformation quality. Thus this requirement is not directly addressed in this thesis, but a way how to compute and how to improve the quality of a transformation is proposed.

VI. Synchronize Long Transformation Chains

*Long Transformation Chains* of several joints must be synchronized.

This requirement is closely related two the latter one, except that the focus is not on synchronizing sensor data with *TF Data*. The focus is on how to synchronize many joints in a chain. This is the case for complex manipulators, where often six or more joints exist. Nevertheless, the problems as well as the solutions are the same as described above.

VII. High Clock Accuracy

The *clocks* of different computers have to be synchronized exactly.

If components for a robot run on more than one PC, the clocks have to be synchronized exactly. This is necessary, because time is one of the main factors in the quality measurement as it will be explained later. High clock synchronicity can be achieved by using the method explained in Section 3.3.2 and thus will not be considered in this thesis.
VIII. Provide Quality Measurement

A Quality Measurement must represent the quality of a transformation at a specific point in time.

One of the most important requirements is to provide a quality measurement for transformations. The quality measurement allows to decide how the system must be configured to obtain the needed accuracy. A good quality measurement in 3D Cartesian space is the position uncertainty of the frame origin.

IX. Separate Server and Client

Server and Client components must be strictly separated through Communication Objects.

It is important to separate the different components. This means that there has to be a clear interface, normally a Communication Object, which describes how the components exchange the TF Data. Thereby the whole information must be included in one message to reduce communication overhead and to avoid unnecessary fusion of different messages. This requirement comes with the use of component-based system, but it is major, because it allows easy reuse and composition of components.

Requirements Addressed in this Thesis

Out of the requirements described above the most important requirements for managing coordinate frame transformations in a distributed and component-based robot system will addressed by this work. Requirements like a high clock accuracy or to publish a message only once are more general requirements not directly related to transformations. Instead these requirements address the framework or middleware which is used to build the component-based system. Thus the requirements must be satisfied by the framework and are not addressed in this thesis, were the focus is on how the complexity of transformations can be handled by the system. Therefor this thesis will look on how good the quality of a transformation is, to enable the developer to see occurring problems and to take appropriate counter-measures. Further to actually understand the system a method to describe the system with its components as well as the data flow between the components is needed.

Therefor this thesis starts by introducing a way to describe such a system on an abstract level. This abstract representation of the system also makes it possible to compare different system implementations. Afterwards a quality measurement for a transformation chain is given. Based on these measurement steps can be taken to achieve the accuracy needed for specific applications. Finally the Communication Object for TF Data which is exchanged by the components is specified.

4.3 Abstract System Description

The first step to manage coordinate frames in a system is to actually understand the system. Thus in this section a set of elements is introduced, that allows to describe a component-based system from the tf view. The tf view includes all elements that are related to coordinate frames, but leaves all the other elements out. Further this abstract description allows to easily compare already existing systems, like the ones presented in Section 2.2 with each other. Additionally also different proposals for a new system can be compared on this level to find the pros and cons of each approach. To actually describe the system five elements are needed:
• **Components** are the building blocks of each system. They send data via ports to other components.

• **TF Data** is the data that contains the information on how to perform the transformations between the frames. One data element always contains the transformation between two tf frames.

• **Joint Data** is for example the angle of a revolute joint or the position in case of a prismatic joint. Out of this and some additional data it is possible to determine the TF Data. For more information on joints and how this conversion works, see Section 3.1.

• **Normal Data** is all other data that components exchange. In relation to transformations this is mostly sensor data.

• **Ports** are the connection points between components and used to exchange data. They can be used in synchronous (request/reply) and asynchronous (publish/subscribe) manner.

The graphical expressions of these elements can be found in Figure 4.7. In this figure also an additional element named *Comment* is introduced. This element does not describe any element of the system, but it allows to add some additional information to a specific element.

Out of these elements it is possible to construct any kind of system. This is done by connecting the ports of different components. Over these links messages containing the specific data are exchanged. A list of how the elements can be combined to form a system and what the corresponding expression means is presented in Figure 4.8. As it can be seen from this figure, it is possible to get data from a component in different ways. Components can convert Joint Data to TF Data with some conversion rules described in the component itself or loaded from a file. Further it is possible to build chains of components, where each component can add the message it has received from another component to the outgoing message.
4.3. ABSTRACT SYSTEM DESCRIPTION

The shown elements provide a method to describe a component-based system on an abstract level. However like introduced in Section 1.2 there also exists a tree representation of the coordinate frames on the robot. This tf tree shows how the frames are connected in a parent/child relationship. To link these two representations corresponding names can be assigned to tf frames and TF Data. How this looks like is shown in Figure 4.9. The names can either be written into the TF Data element or next to it. This labeling allows to see which TF Data contains the information for which tf frame. Further the components providing the data for a tf frame can be determined.
 CHAPTER 4. METHOD

4.4 Quality of Transformations

This section describes how the quality of a single transformation and a transformation chain can be measured. The measurement of the quality allows to decide how the robotic system has to be configured to determine the wanted accuracy. A good quality measurement in space is to use position probabilities, which can be efficiently represented as a 3-dimensional normal distribution.

The basic idea is simple and can be seen as the Prediction Step of the Kalman Filter [16]. According to this, a state vector $x$ and a corresponding covariance matrix $P$ is defined. If the quality of a transformation wants to be known at a specific point in time, the state and covariance must be propagated to this point in time. For a sampled-data system, where continuous time is considered, but only the covariance and state at discrete time instants are interesting, the formulas in [29, p. 111] can be used. There the formula for the propagation of the state $x$ is defined as

$$ x_{k+1} = Ax_k + Bu_k + w_k $$

(4.1)

where $x_{k+1}$ is the state vector at the next time instant. The $n \times n$ matrix $A$ relates the state at the current time step $k$ to the next time step $k + 1$. The matrix $B$ relates the optional control input $u$ to the state $x$ and is not needed in this thesis. The random variable $w_k$ represent the normal distributed process noise. To propagate the covariance matrix the equation

$$ P_{k+1} = AP_kA^T + Q_k $$

(4.2)

is used, where $P_k$ is the current covariance and $P_{k+1}$ is the covariance at the next time step. The matrix $Q_k$ represents the normal distributed process noise like $w_k$ in the previous equation. Together $x$ and $P$ can be seen as a model that can be used to calculate the quality of a transformation.
4.4. QUALITY OF TRANSFORMATIONS

4.4.1 Modelling of Joints

For a hinge joint, which is probably the widest used joint on service robots, the state vector $x$ for a simple model can be defined as

$$x = \begin{bmatrix} s \\ v \end{bmatrix}$$

where

$s$ = angle of the joint

$v$ = angle velocity of the joint

Further the acceleration as well as the change in acceleration could be added to the state vector as further elements. This, of course would build an even more accurate model for propagating the state vector. In the described case the matrix $A$ can be determined out of the formulas for mapping $s$ and $v$ from time step $k$ to $k + 1$. If the angular velocity is considered as constant and the time which past between time step $k$ and $k + 1$ as $T$, the following equations can be formulated:

$$s_{k+1} = s_k + T \cdot v_k$$

$$v_{k+1} = v_k$$

Out of these equations it is easy to determine the matrix $A$ as

$$A = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix}$$

Further the covariance matrix $P$ which represents the uncertainty, can be built out of the single variances $\sigma^2_s$ and $\sigma^2_v$. It is well know that the diagonal of the covariance matrix contains the variance of each single variable, thus

$$P = \begin{bmatrix} \sigma^2_s & 0 \\ 0 & \sigma^2_v \end{bmatrix}$$

In the simplest case where no additional noise nor control input exists the Equations 4.1 and 4.2 for propagating the state vector $x$ and covariance $P$ can be written as

$$x_{k+1} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} s_k \\ v_k \end{bmatrix}$$

$$P_{k+1} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \sigma^2_s & 0 \\ 0 & \sigma^2_v \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

In contrast to the hinge joint which rotates about an axis, the prismatic joint moves along an single axis. Nevertheless the state vector stays the same with only the fact, that $s$ is not representing the angle, but the position. Further $v$ is not representing an angular velocity, but a normal velocity. The matrices $A$ and $P$ stay the same.

4.4.2 Mapping Joint Models to Cartesian Coordinates

The model for a hinge joint described in the previous section works in angular space and thus can not directly be used for frame transformations which are represented in 3D Cartesian space. Therefore the joint model has to be mapped to Cartesian space, which will be done in the following for hinge and prismatic joints. The other joints described in Section 3.1 but not handled here, can be mapped in a similar way.
Hinge Joint

A hinge joint does not have any influence on the position accuracy of the joint frame, because it represents a rotation and not a translation. Thus it only effects the position accuracy of Child frames. The angular uncertainty of the model must therefore be applied when calculating the position uncertainty of the Child frames.

To perform the mapping for a model of a hinge joint to Cartesian space, additionally the distance $d$ must be known. This value describes the distance from the origin of the Child frame to the origin of the joint frame. This distance is measured in the plane perpendicular to the axis of rotation. Together the distance $d$ and the angle $s$ from the model describe a point in polar coordinates. In the following equations, which describe the mapping steps the value $s$ and $\sigma_s^2$ are named $\alpha$ and $\sigma_\alpha^2$. The velocity in the model is only necessary for the propagation of the position and thus not used for the mapping.

The mapping of polar coordinates to Cartesian coordinates can be done by simply using trigonometrical functions. Figure 4.10 visualizes the relationship of polar and Cartesian coordinates, where the point $U$ with polar coordinates $d, \alpha$ is mapped into Cartesian coordinates with $x, y$. The equations for calculating the values of $x$ and $y$ simply are

\[
\begin{align*}
  x &= d \cdot \cos \alpha \\
  y &= d \cdot \sin \alpha
\end{align*}
\]

This relationship can also be seen as a function $F$ which transforms the polar coordinates $[d \ \alpha]^T$ into Cartesian coordinates $[x \ y]^T$. This relationship can be written as

\[
F \left( \begin{bmatrix} d \\ \alpha \end{bmatrix} \right) = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} d \cdot \cos \alpha \\ d \cdot \sin \alpha \end{bmatrix}
\]

The normal distributed uncertainty $\Sigma_{d\alpha}$ can be mapped from polar to Cartesian space with the same function $F$. Since the new uncertainty should also be normal distributed, the mean and variance must satisfy the equations

\[
\begin{bmatrix} x \\ y \end{bmatrix} \sim N \left( \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \Sigma_{xy} \right)
\]

\[
\begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix} = F \left( \begin{bmatrix} \mu_d \\ \mu_\alpha \end{bmatrix} \right) = \begin{bmatrix} \mu_d \cdot \cos \alpha \\ \mu_d \cdot \sin \alpha \end{bmatrix}
\]

The Cartesian covariance matrix $\Sigma_{xy}$ can be calculated out of the polar covariance matrix $\Sigma_{d\alpha}$, by
applying the following equation where $\nabla F$ is the Jacobian matrix of $F$.

$$
\Sigma_{xy} = \nabla F_{d\alpha} \Sigma_{d\alpha} \nabla F_{d\alpha}^T
$$

with

$$
\nabla F_{d\alpha} = \begin{bmatrix}
\frac{\delta f_1}{\delta \alpha} & \frac{\delta f_1}{\delta \alpha} \\
\frac{\delta f_2}{\delta \alpha} & \frac{\delta f_2}{\delta \alpha}
\end{bmatrix} = \begin{bmatrix}
\cos \alpha & -d \sin \alpha \\
\sin \alpha & d \cos \alpha
\end{bmatrix}
$$

$$
\Sigma_{d\alpha} = \begin{bmatrix}
\sigma_x^2 & 0 \\
0 & \sigma_y^2
\end{bmatrix}
$$

The new matrix $\Sigma_{xy}$ is only an approximation, because with $\nabla F$ the derivation at point $[\mu_x \mu_y]^T$ is taken. In Figure 4.11 the standard deviation in the polar and Cartesian system is shown. Here it gets obvious that the mapping is only an approximation, because the uncertainty in the Cartesian space is an ellipse whereas in the polar system it is an arc.

![Figure 4.11: Uncertainty in polar and Cartesian coordinates.](image)

The next step is the mapping of the 2D point to 3D space. This mapping can be done by setting the value for the axis of rotation to zero. This is valid, because a hinge joint always rotates a point in the plane perpendicular to the axis of rotation. Since there are three axis about which the joint can rotate, there are also three cases how the Cartesian coordinates in 2D space have to be mapped to 3D space. The different mappings for the mean $\mu_{xy}$, as well as the covariance $\Sigma_{xy}$ are listed below.

Rotation about x-axis: $\mu_{xyz} = \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}$, $\Sigma_{xyz} = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{yx} & \sigma_{yy} \end{bmatrix}$

Rotation about y-axis: $\mu_{xyz} = \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}$, $\Sigma_{xyz} = \begin{bmatrix} \sigma_{xx} & 0 \\ 0 & \sigma_{yy} \end{bmatrix}$

Rotation about z-axis: $\mu_{xyz} = \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}$, $\Sigma_{xyz} = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{yx} & \sigma_{yy} \end{bmatrix}$
Prismatic Joint

The mapping of a prismatic joint is more straightforward, because the model is already defined in Cartesian space. The only thing that must be done is to set the other two axes, in which the joint is not moving, to zero. Thus the mapping of a prismatic joint with the model \( x_k \), \( P_k \) can be done as

\[
\begin{align*}
\mathbf{x}_k &= \begin{bmatrix} s \\ v \end{bmatrix}, \\
\mathbf{P}_k &= \begin{bmatrix} \sigma^2_s & 0 \\ 0 & \sigma^2_v \end{bmatrix}
\end{align*}
\]

move in x-axis: \( \mu_{xyz} = \begin{bmatrix} s \\ 0 \\ 0 \end{bmatrix}, \quad \Sigma_{xyz} = \begin{bmatrix} \sigma^2_s & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \)

move in y-axis: \( \mu_{xyz} = \begin{bmatrix} 0 \\ s \\ 0 \end{bmatrix}, \quad \Sigma_{xyz} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \sigma^2_s & 0 \\ 0 & 0 & 0 \end{bmatrix} \)

move in z-axis: \( \mu_{xyz} = \begin{bmatrix} 0 \\ 0 \\ s \end{bmatrix}, \quad \Sigma_{xyz} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & \sigma^2_s \\ 0 & 0 & 0 \end{bmatrix} \)

4.4.3 Quality of Transformation Chains

Until now only the quality of a single transformation was considered. This section will show how single position uncertainties can be merged to describe the uncertainty of a transformation chain. To do this, the simple \( tf \) chain shown in Figure 4.12 is used. This chain consists of a root frame called \textit{base\_frame} and two other frames, which are linked through \textit{prismatic joints}. The joint that links the \textit{base\_frame} with the \textit{x\_frame} moves in the x-direction whereas the joint linking the \textit{x\_frame} with the \textit{y\_frame} moves in y-direction. As an example the transformation \( M \), as well as the position uncertainty \( \Sigma \) can be defined as:

- \textit{base\_frame} → \textit{x\_frame}:

\[
\begin{align*}
M_a &= \begin{bmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \\
\Sigma_a &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.001 & 0 \\ 0 & 0 & 0 \end{bmatrix}
\end{align*}
\]
• $\text{x\_frame} \rightarrow \text{y\_frame}$:

\[
M_b = \begin{bmatrix}
0.7071 & -0.7071 & 0 & 1 \\
0.7071 & 0.7071 & 0 & 2 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}, \quad \Sigma_b = \begin{bmatrix}
0.001 & 0 & 0 \\
0 & 0.5 & 0 \\
0 & 0 & 0
\end{bmatrix}
\]

The uncertainties are given in their local frame and look like shown in Figure 4.13. There it can be seen that the uncertainty for the $\text{x\_frame}$ is in x-direction and for the $\text{y\_frame}$ it is in y-direction. To merge these uncertainties, to get an overall quality for the chain, the single variances have to be converted into a single reference frame. This frame is the root of the chain and would be the $\text{base\_frame}$ in the example.

![Figure 4.13: Single uncertainties for the frames.](image)

To change the coordinate frame of a covariance matrix $\Sigma$ the equation

\[
\Sigma^+ = R \Sigma R^T
\]

can be used, where $R$ is the Rotation of the transformation. The rotation for the $\text{x\_frame}$ would be $R(M_a)$ whereas for the $\text{y\_frame}$ it would be $R(M_a M_b)$.

After mapping the variances to a single reference frame, the next and last step is to merge them. The result of this step is the position uncertainty for the last frame of the chain in relation to the root frame of the chain. Thus the uncertainty $\Sigma_{ab}$ describes the quality of the whole transformation chain. It can simply be calculated by taking the sum of the single covariance matrices. For the example this would be

\[
\Sigma_{ab} = \Sigma_a + \Sigma_b.
\]

The resulting $\text{tf chain}$ with the single uncertainties $\Sigma_a$, $\Sigma_b$ and the total uncertainty $\Sigma_{ab}$ is shown in Figure 4.14. In this figure it gets obvious, that the single uncertainties add up to a bigger uncertainty. This of course makes sense, because each joint introduces some additional uncertainty into the chain, which must increase the overall uncertainty.

If a $\text{hinge joint}$ is included in the chain it is not enough to determine the position uncertainty of the $\text{Child frame}$ and to add it to the overall position uncertainty. This is due to the fact that the
position uncertainty increases with the distance of the Child frame to the joint frame. To get the right uncertainty the distance \(d\) must be the distance between the joint frame and the frame for which the overall uncertainty should be calculated. The distance \(d\) is the one used for converting the joint model to Cartesian space. If the uncertainty is determined in this way, the overall uncertainty can be calculated as described above.

4.4.4 Position Prediction vs. Publishing Frequency

This section gives insights into how the publishing frequency should be chosen to get the wanted accuracy and which information should be included in the state vector of the quality measurement. As shown in Section 4.4, it is possible to predict the position of the joint with the values included in the state vector. Until now the only values in the state vector were the current position \(s\) and the velocity \(v\) which formed the vector \([s \ v]^T\). It is for example also possible to only use the position \([s]\) or to additionally add the acceleration which gives the vector \([s \ v \ a]^T\). If more information like \(v\) and \(a\) are included, the prediction of the position is more accurate, but with the drawback that more data has to be exchanged. A good tradeoff therefor is to use \(s\) and \(v\), because in most cases these two values are easily available and they allow a far better prediction than only using \(s\). If only \(s\) is used, it is just possible to increase the uncertainty for any point in time. The position itself stays the same, because the direction in which the joint moves is unknown. Regardless of how good the values in the state vector are reflecting the reality, the uncertainty will always increase the bigger the period of time used for predicting the position is.

After setting the state vector to \([s \ v]^T\), the only way to improve the quality is to increase the frequency at which the TF Data is captured. A higher frequency just means that the period of time for predicting the position is decreased. Thus, if the position quality should be better than a specific
value, the frequency $f = \frac{1}{T}$ at which the TF Data has to be captured can be calculated as

$$P_{k+1} = A P_k A^T = \begin{bmatrix} \sigma^2_{s,k+1} & 0 \\ 0 & \sigma^2_{v,k+1} \end{bmatrix} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \sigma^2_{s,k} & 0 \\ 0 & \sigma^2_{v,k} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ T & 1 \end{bmatrix}$$

From this it is possible to write

$$\sigma^2_{s,k+1} = \sigma^2_{s,k} + T^2 \sigma^2_{v,k}$$

$$T^2 = \frac{\sigma^2_{s,k+1} - \sigma^2_{s,k}}{\sigma^2_{v,k}}$$

This equation allows to calculate the frequency, at which the TF data must be captured, for an arbitrary worst case position quality of $\sigma^2_{s,k+1} = 0.1^2 \text{ rad}$. To do this also the values for $\sigma^2_{s,k}$ and $\sigma^2_{v,k}$ have to be known. An assumption for this example is $\sigma^2_{s,k} = 0.01^2$ and $\sigma^2_{v,k} = \left(\frac{\pi}{4}\right)^2$. From these values the frequency can be calculated as

$$T^2 = \frac{\sigma^2_{s,k+1} - \sigma^2_{s,k}}{\sigma^2_{v,k}}$$

$$= \frac{0.1^2 - 0.01^2}{\left(\frac{\pi}{4}\right)^2}$$

$$= 0.016$$

$$\Rightarrow T = 0.1267 \text{ sec}$$

$$f = \frac{1}{T} = 7.89 \text{ Hz}$$

From this calculation follows, that the TF Data must be captured with a frequency of at least 7.89 Hz to achieve the desired quality. In reality and with some safety this normally results in a frequency of 8 to 10 Hz.

### 4.5 Client/Server Separation

As known from component-based systems and also stated in the requirements in Section 4.2, it is important to separate components in a clear way. This separation allows to divide the overall system complexity into small parts, which can be implemented by components. Out of these components complex systems can be built.

The separation of components is done by defining the messages exchanged between the components. These messages are called Communication Objects or in short CommObjects. CommObjects can either be plain data structures like they are in ROS or they can specify additional access methods like it is in SMARTSOFT. These additional methods allow operations on the contained data like conversion or filtering. In contrast thereto in ROS additional methods in special libraries are provided. Both approaches have pros and cons, but in general they provide the same capabilities and thus it is not distinguished between them in this work.
In the following the values that must be exchanged by the components to actually do coordinate frame transformations are elaborated. Finally a proposal for a Communication Object with all its data is made.

### 4.5.1 Validity of TF Data

One requirement specified in Section 4.2 is to consider the difference between static and dynamic frames. The main difference between them is in the quality measurement and in the period of time the TF Data for a frame is valid. The validity defines how long received TF Data can be used to still give useful results.

To answer this question two components $A$ and $B$ are considered. Component $A$ continuously publishes $tf$ messages at a specific frame rate. A $tf$ message is a Communication Object containing TF Data. The published messages are received by component $B$ that has subscribed for them. Further, component $B$ wants to transform the laser scan it gets from another component $C$ with the received TF Data from component $A$. To do this, the laser scan and a corresponding $tf$ message has to be matched. This can be done by using the last received $tf$ message or by looking for the best matching $tf$ message according to the time stamps.

The best match according to the time stamp can be defined in different ways. First the best matching $tf$ message could be the last one captured before the laser scan. In Figure 4.15 this would be the TF Data at time $t_{n+1}$. It is also possible to say, that the next TF Data captured after the laser scan should be used. This then would be $t_{n+2}$. Finally the best matching TF Data could be the one, that has the smallest time difference to the laser scan. In the example this would also be $t_{n+2}$.

As shown, there are different ways to define the best matching TF Data. The probably most convenient one is to use the TF Data where the difference between the TF Data and the laser scan is the smallest. Thus the validity of a $tf$ message can be defined as $t_{\text{valid}} = t \pm \frac{\text{freq}}{2}$, where $t$ is the point in time at which the TF Data was captured.

In reality the frequency, at which TF Data is captured, can jitter. Thus, if the validity defined above is used, situations with gaps and overlaps in the range of valid TF Data can occur. Such a situation is shown in Figure 4.16. The gaps result in problems, because it can happen that there is no valid TF
Data for a specific point in time. Thus it is not possible to perform the wanted transformation. To solve this problem the validity of the TF Data has to be increased by some factor. As an example $t_{\text{valid}}$ could be set to $t_{\text{valid}} = t \pm \frac{\text{freq}^{-1}}{1.5}$. This then results in a case like shown in Figure 4.17 where no gaps, but overlaps exist. There is no specific rule how long $t_{\text{valid}}$ has to be. It should be as short as possible, but it is important that there are no gaps in the validity of TF Data. In contrast to gaps, overlaps are not a big problem, because it only means that there are two valid tf messages for a point in time. In this case the best matching tf message can be used. The considerations until now only covered dynamic tf frames in the moving state. Thus the next step is to define the validity of tf messages for static frames and for dynamic tf frames in the fixed state.

The validity of static frames can easily be defined as $t_{\text{valid}} = t + \infty$, because it is known that the transformation never changes over time. Thus it is valid from now until forever.

The fixed state of dynamic frames is catchier. It is known that the transformation of a fixed frame stays the same until the joint moves again. Thus $t_{\text{valid}}$ could be specified as $t_{\text{valid}} = t + t_{\text{until moving state}}$. The problem of course is, that the time until the joint starts to move is unknown. It could be any number in the range of $0 < t_{\text{until moving state}} \leq \infty$. To solve this problem different approaches can be used. The first approach is to specify $t_{\text{valid}}$ in the same way as it is done in the moving state, which is $t_{\text{valid}} = t \pm \frac{\text{freq}^{-1}}{1.5}$. Thus the TF Data must be resent with the same frequency as in the moving state. This results in a waste of resources, especially since the joints normally are in fixed state the most time. The second approach is to set $t_{\text{valid}} = t + \infty$, like it is done for static frames. The TF Data then would be valid until a newer tf message is received. If there is valid TF Data both for the fixed state and the moving state, as shown in Figure 4.18, the data for the fixed state should be used. This is the better choice, because it is known that the joint has started to move at the time of the new tf message.

As shown, validity for TF Data can be specified for all cases, whereas $t_{\text{valid}}$ differs for the different cases. It is also possible to work without $t_{\text{valid}}$. This can be done, if the message delivery is reliable and if the framework ensures the frequency at which TF Data is published. If the frequency is not reached the framework must raise an error that invalidates the last received tf message from this frame. The only thing that still has to be done is to distinguish the different states of joints.
CHAPTER 4. METHOD

4.5.2 Specification of the Communication Object

To do coordinate frame transformations various values must be transferred from the server component to the client component. For transmitting this data a Communication Object is used, which contains all the needed information to perform the transformation between two frames. The separation of components by Communication Objects allows to easily reuse components in other system setups.

To describe a coordinate frame in 3D Cartesian space, the translation as well as the rotation have to be specified. The translation can simply be defined by a vector of three values \([x \ y \ z]^T\). The rotation as known from Section 3.2.2, where Euler Angles were introduced, can be defined by at least three values. However, the better way to specify a rotation is to use Quaternions (see Table 3.1) which consist of four values \([q_0 \ q_1 \ q_2 \ q_3]^T\). Together these values allow to describe any coordinate frame in 3D space.

Further to build a transformation chain the parent frame and the name of the current frame have to be specified. Figure 4.19 shows a simple tf tree with two frames. If the current frame is the laser_frame, the parent frame must be set to robot_frame. This parent/child relationship allows to reconstruct the tf tree on the client site from different tf messages. Further they allow to specify the wanted transformation by giving the source and target frame. The Communication Object or some other Class in a library can then look for a path through the tf tree and if there is one, the resulting transformation is returned. If no path could be found an error is raised.

![Figure 4.19: Simple tf tree with two frames.](image)

Also a good idea, but not absolutely necessary, is to define a sequence id which is just increased by one each time a message is sent. With this id it is easy to distinguish different messages and to bring them into a sequence.

If the quality measurement introduced in Section 4.4 should also be added, the timestamp at which the TF Data was captured, as well as the model for propagation must be included. The model consists of the state vector, the covariance matrix and the matrix \(A\) which relates the state at time \(k\) with the state at \(k + 1\). The matrix \(A\) must not be sent in the message, because it is constant. The joint type (revolute, prismatic, . . . ) has to be specified, because the mapping of the state vector and the covariance matrix to 3D Cartesian space depends on this type. Further, the size of the state vector as well as the size of the covariance matrix is also different for different types of joints. Additionally the axis about which the revolute joint rotates or in which the prismatic joint is moving has to be specified. In case of a planar joint the plane in which the movement happens has to be specified. The axis or plane can be defined by a 3D vector which represents the rotation axis for a revolute joint, the direction of movement for a prismatic joint or the plane normal for a planar joint. These additional values in the CommObject allow to calculate the quality of a transformation independently of any other component.

The last values needed in the Communication Object come from the previous section, where the
validity of TF Data was introduced. Three values, namely the point in time \( t \) at which the TF Data was captured, the period of time \( t_{\text{valid}} \) how long the TF Data is valid and the type of the transformation have to be added. Since the timestamp needed for the quality measurement is equal to \( t \), the only values that have to be added are \( t_{\text{valid}} \) and the type of the transformation.

All the values described until now define the Communication Object which in pseudo code could look like shown in Listing 4.1.

Listing 4.1: Values for the Communication Object

```plaintext
float64 x, y, z            # 3D vector
float64 q0, q1, q2, q3     # Quaternion
string parent_frame        # name of the parent frame
string current_frame       # name of the current frame
uint32 seq_id              # optional sequence number
time timestamp             # time when TF Data was captured
float64[] state_vector     # state vector of size n
float64[][] covariance_matrix # covariance matrix of size n x n
enum joint_type            # type of the joint
float64 jx, jy, jz         # 3D vector specifying joint axis
time time_valid            # period of time, the data is valid
enum frame_type            # dynamic (fixed/moving) or static frame
```

### 4.6 Constructing TF Systems

#### 4.6.1 System Modelling

In this section some real world examples with the elements described in Section 4.3 will be presented. These examples show that the introduced elements allow to model any kind of system, since they provide a representation for any relevant element of the system.

For the next examples a setup where a laser ranger is mounted on an arm is considered. The arm is statically mounted on a mobile robot base. The components which are involved are base, arm, laser and another component named \( xy \). The latter component is interested in the transformation chain of the laser ranger and the laser scan published by the laser component. The corresponding tf tree can be seen in Figure 4.20. First the modelling of this setup with the ROS tf system is shown.

![TF tree for the example setup.](image-url)
In ROS components can convert Joint Data to TF Data by their own or they can send the Joint Data to a global component called robot_state_publisher. This component then converts the Joint Data to TF Data according to the rules described in an URDF file. After doing this the TF Data is transmitted to all components interested in tf. The two different ways how the example can be modeled are shown in Figure 4.21.

![Figure 4.21: Modelling of the ROS tf system.](attachment:image.png)

Another framework that comes with partial tf support is SMARTSOFT. It has no explicit build in support for transformations but the current implementation allows to build chains of components, that look similar to the tf tree. The resulting model is shown in Figure 4.22. As it can be seen from the

![Figure 4.22: Modelling of the current tf support in SMARTSOFT.](attachment:image.png)

model, in SMARTSOFT the base component itself converts the Joint Data to the TF Data and sends it to the next component which is the arm component. This component does the same, but adds the TF Data from the base to its own data. Finally the laser component sends the laser scan and the received TF Data encapsulated in one CommObject to the interested component xy.

There can, of course, be other implementations of systems. One for example would be a central component, managing all actuators. Each component which is then interested in a transformation has to ask this central component for the Joint Data. The xy component then calculates the transformation out of the received Joint Data. The model in this way would look like shown in Figure 4.23.

The shown models are only a small subset of all possible implementations, but they showed that with the proposed elements it is possible to model all kind of different systems. Thus it is possible to easily compare different systems. Further the data flow of tf messages in these systems can be examined and allows to find bottlenecks and single point of failures in the system. Such a single point of failure for example would be the robot_state_publisher in Figure 4.21(b).
4.6. **CONSTRUCTING TF SYSTEMS**

4.6.2 Calculating the Quality of a Joint

In Section 4.4 a quality measurement based on probabilities was defined and a method to convert the joint model to Cartesian space was shown. To make this process clearer, the calculations needed to perform the mapping for a simple revolute joint are shown in the following.

For this example, it is assumed that the joint starts at $s_0 = 0 \text{ rad}$ and is currently moving with $v_0 = \frac{\pi}{2} \text{ rad/sec}$. The variance in the angle is $\sigma^2_{s,0} = 0.01^2$ and the variance in velocity is $\sigma^2_{v,0} = (\frac{\pi}{4})^2$. This results in

$$x_0 = \begin{bmatrix} s_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{\pi}{2} \end{bmatrix}$$

$$P_0 = \begin{bmatrix} \sigma^2_{s,0} & 0 \\ 0 & \sigma^2_{v,0} \end{bmatrix} = \begin{bmatrix} 0.01^2 & 0 \\ 0 & (\frac{\pi}{4})^2 \end{bmatrix}$$

This allows the propagation of the state vector $x$ and the covariance matrix $P$ to a further point in time $t_1 = 0.1 \text{ sec}$, with Equation 4.1 and Equation 4.2 as

$$x_1 = Ax_0$$

$$= \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} s_0 \\ v_0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0.1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ \frac{\pi}{2} \end{bmatrix}$$

$$= \begin{bmatrix} 0.1571 \\ 1.5708 \end{bmatrix}$$

$$P_1 = AP_0A^T$$

$$= \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \sigma^2_{s,0} & 0 \\ 0 & \sigma^2_{v,0} \end{bmatrix} \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix}^T$$

$$= \begin{bmatrix} 1 & 0.1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0.0001 & 0 \\ 0 & 0.6169 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0.1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.0063 & 0.0617 \\ 0.0617 & 0.6169 \end{bmatrix}$$

The matrix $P_1$ now gives the quality of the transformation at time $t_1$. As it can be seen, the uncertainty of the velocity $\sigma^2_v$ did not change. This is the case because there is no factor which influences the velocity. In contrast thereto the uncertainty of the angular position $\sigma^2_s$ changes from $\sigma^2_{s,0} = 0.0001$ to $\sigma^2_{s,1} = 0.0063$. 

![Figure 4.23: Model where a central component manages all actuators.](image-url)
The prediction until now was done using the *Joint Model*. To get Cartesian coordinates the mapping described in Section 4.4.2 has to be applied, which is

\[
\begin{bmatrix}
x \\
y \\
\end{bmatrix} = \begin{bmatrix}
d \cdot \cos \alpha \\
d \cdot \sin \alpha \\
\end{bmatrix}
\]

\[
\Sigma_{xy} = \nabla F_{dx} \Sigma_{d\alpha} \nabla F_{da}^T
= \begin{bmatrix}
\cos \alpha & -d \sin \alpha \\
\sin \alpha & d \cos \alpha \\
\end{bmatrix}
\begin{bmatrix}
\sigma_d^2 & 0 \\
0 & \sigma_\alpha^2 \\
\end{bmatrix}
\begin{bmatrix}
\cos \alpha & -d \sin \alpha \\
\sin \alpha & d \cos \alpha \\
\end{bmatrix}^T
\]

The value for \(\sigma_d^2\) is 0 because the distance is fixed. The value for \(\sigma_\alpha^2\) can be taken from \(P_k\) and is equal to \(\sigma_s^2\). For this example a coordinate frame, which position should be calculated, has the distance \(d = 4\, \text{cm}\) from the joint. Thus the position of the coordinate frame at time \(t_0 = 0\, \text{sec}\) and \(t_1 = 0.1\, \text{sec}\) can be calculated as

- \(t_0 = 0\, \text{sec}\)

\[
\begin{bmatrix}
x_0 \\
y_0 \\
\end{bmatrix} = \begin{bmatrix}
4 \cdot \cos 0 \\
4 \cdot \sin 0 \\
\end{bmatrix} = \begin{bmatrix}
4 \\
0 \\
\end{bmatrix}
\]

\[
\Sigma_{xy_0} = \begin{bmatrix}
\cos 0 & -4 \sin 0 \\
\sin 0 & 4 \cos 0 \\
\end{bmatrix}
\begin{bmatrix}
0 & 0.01^2 \\
0 & 0.01^2 \\
\end{bmatrix}
\begin{bmatrix}
\cos 0 & -4 \sin 0 \\
\sin 0 & 4 \cos 0 \\
\end{bmatrix}^T
= \begin{bmatrix}
0 & 0 \\
0 & 0.0016 \\
\end{bmatrix}
\]

- \(t_1 = 0.1\, \text{sec}\)

\[
\begin{bmatrix}
x_1 \\
y_1 \\
\end{bmatrix} = \begin{bmatrix}
4 \cdot \cos(0.1571) \\
4 \cdot \sin(0.1571) \\
\end{bmatrix} = \begin{bmatrix}
3.9507 \\
0.6258 \\
\end{bmatrix}
\]

\[
\Sigma_{xy_1} = \begin{bmatrix}
\cos(0.1571) & -4 \sin(0.1571) \\
\sin(0.1571) & 4 \cos(0.1571) \\
\end{bmatrix}
\begin{bmatrix}
0 & 0.01^2 \\
0 & 0.01^2 \\
\end{bmatrix}
\begin{bmatrix}
\cos(0.1571) & -4 \sin(0.1571) \\
\sin(0.1571) & 4 \cos(0.1571) \\
\end{bmatrix}^T
= \begin{bmatrix}
0.0025 & -0.0156 \\
-0.0156 & 0.0983 \\
\end{bmatrix}
\]

This result can be visualized, as shown in Figure 4.24. The green lines go from the origin to the mean position of the frame and the blue ellipses represent the variance at the fraction 0.9 of probability mass. The ellipses look rather like lines, because the variance for the distance \(d = 0\). Further the increase in uncertainty can be seen at a glance in this visual representation.

The example showed that the propagation of mean and variance, as well as the mapping to Cartesian space works. Further the probabilistic approach for describing the quality of a transformation is a good choice, because it allows an interpretation in space.
4.6. CONSTRUCTING TF SYSTEMS

4.6.3 Calculating the Quality of a Transformation Chain

In the last example, the quality of a single transformation was considered. This section will show how the single qualities of each element in a chain can be incorporated to get the quality of the whole transformation chain. As an example, the *tf tree* shown in Figure 4.25(a) will be used. This tree consists of a frame called `base_frame` which is the root element of the *tf tree*. The `slide_frame` is linked with the `base_frame` by a *prismatic joint*. This *prismatic joint* can move in the direction of the *x-axis* of the `slide_frame`. The `pan_frame` is linked with the `slide_frame` through a *revolute joint*, that rotates around the *z-axis*. Further the `tilt_frame` is linked with the `pan_frame` through a *revolute joint*, which rotates around the *y-axis*. Finally the `sensor_frame` is linked with the `tilt_frame` through a *fixed joint*. These frames and their direction of movement are shown in Figure 4.25(b). To actually calculate the quality of the transformation chain, assumptions for different values have to be made.

![Figure 4.24: Mean and covariance for $t_0$ and $t_1$.](image)

![Figure 4.25: *Tf tree* and frames for a transformation chain.](image)
In the example the default rotation (see Section 3.1) is considered to be zero for all frames. The vector of the default translation $o$, as well as the values for the state vector $x$ and the covariance matrix $P$ are listed below for each frame. The default translation is just called Translation and describes the translation between the Parent and Child frame when the joint is in the default position. The state vector consists of the current position $s$ and velocity $v$. In the case of a revolute joint the position is the angle in rad and in the case of a prismatic joint, it is the distance in cm from the default position. The state vector and covariance matrix, thus are defined as

$$
x = \begin{bmatrix} s \\
v \end{bmatrix}
$$

$$
P = \begin{bmatrix} \sigma_s^2 & 0 \\
0 & \sigma_v^2 \end{bmatrix}
$$

The point in time where the different TF Data is captured and the point in time, at which the transformation wants to be known, is shown in Figure 4.26.

The default translation, the state vector at time $t_0$ and the covariance matrix at time $t_0$ for the different frames of the tf tree are:

- **slide_frame**:

$$
o_a = \begin{bmatrix} 5 \\
2 \\
8 \end{bmatrix}, \quad x_{a,0} = \begin{bmatrix} 0 \\
100 \\
0 \end{bmatrix}, \quad P_{a,0} = \begin{bmatrix} 0.1^2 & 0 \\
0 & 10^2 \end{bmatrix}
$$

- **pan_frame**:

$$
o_b = \begin{bmatrix} 9 \\
0 \\
3 \end{bmatrix}, \quad x_{b,0} = \begin{bmatrix} 0 \\
\frac{\pi}{2} \\
\frac{\pi}{4} \end{bmatrix}, \quad P_{b,0} = \begin{bmatrix} 0.01^2 & 0 \\
0 & \left(\frac{\pi}{4}\right)^2 \end{bmatrix}
$$

- **tilt_frame**:

$$
o_c = \begin{bmatrix} 1 \\
0 \\
2 \end{bmatrix}, \quad x_{c,0} = \begin{bmatrix} 0 \\
\frac{\pi}{3} \\
\frac{\pi}{4} \end{bmatrix}, \quad P_{c,0} = \begin{bmatrix} 0.01^2 & 0 \\
0 & \left(\frac{\pi}{4}\right)^2 \end{bmatrix}
$$
4.6. CONSTRUCTING TF SYSTEMS

• sensor_frame:

\[ a_d = \begin{bmatrix} 2 \\ 0 \\ 3 \end{bmatrix} \]

After the specification of the values, the next step is to calculate the quality for each single frame. To do this, first all state vectors and covariance matrices are propagated to the desired point in time which is \( t = 300 \text{ ms} \).

• slide_frame: \( T = 300 \text{ ms} - 100 \text{ ms} = 0.2 \text{ s} \)

\[ x_{a,1} = \begin{bmatrix} 1 & 0.2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 100 \end{bmatrix} = \begin{bmatrix} 20 \\ 100 \end{bmatrix} \]
\[ P_{a,1} = \begin{bmatrix} 1 & 0.2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0.1^2 & 0 \\ 0 & 10^2 \end{bmatrix} \begin{bmatrix} 1 & 0.2 \end{bmatrix}^T = \begin{bmatrix} 4.01 & 20 \\ 20 & 10^2 \end{bmatrix} \]

• pan_frame: \( T = 300 \text{ ms} - 235 \text{ ms} = 0.065 \text{ s} \)

\[ x_{b,1} = \begin{bmatrix} 1 & 0.065 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ \frac{\pi}{2} \end{bmatrix} = \begin{bmatrix} 0.1021 \\ \frac{\pi}{2} \end{bmatrix} \]
\[ P_{b,1} = \begin{bmatrix} 1 & 0.065 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0.01^2 & 0 \\ 0 & (\frac{\pi}{2})^2 \end{bmatrix} \begin{bmatrix} 1 & 0.065 \end{bmatrix}^T = \begin{bmatrix} 0.0027 & 0.0401 \\ 0.0401 & (\frac{\pi}{4})^2 \end{bmatrix} \]

• tilt_frame: \( T = 300 \text{ ms} - 180 \text{ ms} = 0.12 \text{ s} \)

\[ x_{c,1} = \begin{bmatrix} 1 & 0.12 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ \frac{\pi}{3} \end{bmatrix} = \begin{bmatrix} 0.1257 \\ \frac{\pi}{3} \end{bmatrix} \]
\[ P_{c,1} = \begin{bmatrix} 1 & 0.12 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0.01^2 & 0 \\ 0 & (\frac{\pi}{3})^2 \end{bmatrix} \begin{bmatrix} 1 & 0.12 \end{bmatrix}^T = \begin{bmatrix} 0.009 & 0.074 \\ 0.074 & (\frac{\pi}{3})^2 \end{bmatrix} \]

After propagating the state vector and the covariance matrix, the next step is to map them into 3D Cartesian space (see Section 4.4). The results are then the single transformations, with the transformation \( M \) and the variance in translation \( \Sigma \) between the origins of the frames. To keep this example compact, only the results after the mapping are listed, which are

• base_frame \( \rightarrow \) slide_frame:

\[ M_a = \begin{bmatrix} 1 & 0 & 0 & 25 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 8 \\ 0 & 0 & 0 & 1 \end{bmatrix} \], \quad \Sigma_a = \begin{bmatrix} 4.01 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \]
• slide_frame $\rightarrow$ pan_frame:

$$M_b = \begin{bmatrix} 0.9947 & -0.1019 & 0 & 9 \\ 0.1019 & 0.9947 & 0 & 0 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \Sigma_b = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

• pan_frame $\rightarrow$ tilt_frame:

$$M_c = \begin{bmatrix} 0.9921 & 0 & 0.1253 & 1 \\ 0 & 1 & 0 & 0 \\ -0.1253 & 0 & 0.9921 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \Sigma_c = \begin{bmatrix} 0 & -0.0003 & 0 \\ -0.0003 & 0.0027 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

• tilt_frame $\rightarrow$ sensor_frame:

$$M_d = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \Sigma_d = \begin{bmatrix} 0.0002 & 0 & -0.0015 \\ 0 & 0 & 0 \\ -0.0015 & 0 & 0.0115 \end{bmatrix}$$

The above uncertainties describe the position uncertainty of the Child frame relative to the Parent frame. To get the overall uncertainty $\Sigma_f$ of the sensor_frame relative to the base_frame all single variances in the chain $\Sigma^{-}$ must be summed up. The variances $\Sigma_a$ and $\Sigma_b$ can directly be used for the variance of the chain. The variances introduced by the hinge joints must be calculated as describe in Section 4.4.2. In particular this means that the uncertainties $\Sigma_c^{-}$ and $\Sigma_d^{-}$ for the sensor_frame must be calculate relative to the pan_frame and tilt_frame. After doing this, the variances $\Sigma_{a...d}^{-}$ must be converted into a single frame. The most suitable frame therefore is the base_frame. The conversion is done with the rules described in Section 4.4.3. The final variance $\Sigma_f$ which consists off all summed up single variances after conversion $\Sigma_{a...d}^{+}$ can be calculated as

$$\Sigma_f = \Sigma_a^+ + \Sigma_b^+ + \Sigma_c^+ + \Sigma_d^+ = \begin{bmatrix} 4.0100 & -5.647 \cdot 10^{-4} & -5.093 \cdot 10^{-5} \\ -5.647 \cdot 10^{-4} & 2.611 \cdot 10^{-3} & 3.258 \cdot 10^{-5} \\ -5.093 \cdot 10^{-5} & 3.258 \cdot 10^{-5} & 1.169 \cdot 10^{-2} \end{bmatrix}$$

The final transformation $M_f$ from the base_frame to the sensor_frame can be computed by multiplying the single transformation matrices $M_a...d$. Thus $M_f$ is calculated as

$$M_f = M_a M_b M_c M_d = \begin{bmatrix} 0.9869 & -0.1019 & 0.1247 & 37.3428 \\ 0.1011 & 0.9947 & 0.0127 & 2.3424 \\ -0.1253 & 0 & 0.9921 & 15.7256 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
4.6. CONSTRUCTING TF SYSTEMS

This homogeneous matrix can also be represented as a translation vector and a rotation described with an zyx Euler sequence.

\[
\begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix} = \begin{bmatrix}
  37.3428 \\
  2.3425 \\
  15.7256
\end{bmatrix} \text{ cm}, \quad \begin{bmatrix}
  \phi_x \\
  \theta_y \\
  \psi_z
\end{bmatrix} = \begin{bmatrix}
  0 \\
  7.20 \\
  5.85
\end{bmatrix} \text{ deg}
\]

The single frames and the summed up position uncertainty of the frame origin is shown in Figure 4.27. The printed ellipses are $3\sigma$ ellipses. It can be seen that the slide_frame and the pan_frame do not have an ellipse, because these frames only have an uncertainty in one dimension. The tilt_frame has an 2D ellipse, because the pan_frame has added some uncertainty in $y$-direction. Finally the sensor_frame has an 3D ellipsoid.

![Figure 4.27: Tf chain with 3σ uncertainty ellipses.](image)

From the values above, as well as from the figure it is obvious, that the uncertainty of the slide_frame has the most impact on the overall position uncertainty. Thus, if the quality should be improved, the best point to start at, is the slide_frame.
Chapter 5

Results

To better examine the methods presented in Chapter 4 a simulation framework was developed which allows the modelling of component-based systems with the elements presented in Section 4.3. The framework was implemented using the Mobile Robot Programming Toolkit (MRPT)\(^1\) which provides methods to do matrix calculation and to visualize 3D data.

For the simulation a model of a robot with an 4DOF (degrees of freedom) arm, a torso that can move up and down, a tilting laser and a head which can tilt and pan is chosen. The corresponding \textit{tf tree} for this robot with all frames is shown in Figure 5.1. In the tree, the \textit{dynamic transformations} are colored orange whereas \textit{static transformations} are colored green. It can be seen, that most transformations are \textit{dynamic} as it is on a real robot.

\begin{figure}[
\centering
\includegraphics[width=\textwidth]{tf_tree.png}
\caption{\textit{Tf tree} of a robot with different actuators and sensors.}
\end{figure}

\(^1\)http://www.mrpt.org/
After defining the setup of the robot, the next step is to look at the component-based system and the components used for the application running on the robot. For the simulation the system shown in Figure 5.2 is used. There are different components, where each component is responsible for one device like the robot arm or a camera. The TF Data which a component provides is labeled with a letter that corresponds to a tf frame in the tf tree. It can be seen, that some components like the head component do provide more than one TF Data. To publish the static transformations a special component named static transforms is implemented. The use of this special component keeps other components which provide sensor data like the laser component free from transformations. Of course it would also be possible to publish the static transformations by the components which provide sensor data.

Further the system contains three components which use the TF Data and the Sensor Data provided by the other components to fulfill tasks like collision avoidance, creating a colored 3D point cloud or a 3D scan. The components achieve this by orchestrating other components. Control messages like a message telling the head to pan left are not shown in the figure, because they are not interesting for doing transformations.

![Component model with data flow. TF Data is linked by names with the tf tree.](image)

5.1 Simulation

For the actual simulation the data flow for the collision avoidance component is used. Thus the static transforms, the torso, the tilt, the laser and the collision avoidance component have to be modeled. After knowing the components the tf chain, describing the transformation needed to transform the laser scan captured in the laser_frame into the base_frame, can be determined in the tf tree. This is done by looking for the path from the base_frame to the laser_frame. The result is the tf chain: base_frame → torso_frame → laser_tilt_frame → laser_frame. For the application of collision avoidance the position accuracy of the laser_frame and in particular of a point captured by the laser ranger is of interest. Thus the simulation should answer the following questions.

How good is the quality at the laser_frame and a point captured by the laser ranger if
5.1. SIMULATION

- the torso_frame as well as the laser_tilt_frame is moving.
- only the laser_tilt_frame is moving.
- only the torso_frame is moving.
- no frame is moving.
- the torso and tilt components run at 10 or 20 Hz.

To actually run the simulation further information about the default offset between the frames and the movement of the joints which connect the frames are needed. Thus for the simulation the following values are assumed where distances are in meter and angles are in radian:

- base_frame $\rightarrow$ torso_frame:
  - translation: $o_{tf} = [0 \ 0 \ 0.5]^T$
  - start position: $p_{tf} = 0$ m
  - velocity: $v_{tf} = 0.4$ m/s
  - position uncertainty: $\sigma_{p,tf}^2 = 0.002^2$
  - velocity uncertainty: $\sigma_{v,tf}^2 = 0.1^2$
  - move axis: $[0 \ 0 \ 1]^T$
  - range: $0.4 \leq p_{tf} \leq 1.4$

- torso_frame $\rightarrow$ laser_tilt_frame:
  - translation: $o_{ltf} = [0.2 \ 0 \ 0.1]^T$
  - start position: $p_{ltf} = 0$ rad
  - velocity: $v_{ltf} = \frac{\pi}{2}$ rad/s
  - position uncertainty: $\sigma_{p,ltf}^2 = 0.01^2$
  - velocity uncertainty: $\sigma_{v,ltf}^2 = \left(\frac{\pi}{4}\right)^2$
  - rotation axis: $[0 \ 1 \ 0]^T$
  - range: $-\frac{\pi}{8} \leq p_{ltf} \leq \frac{\pi}{8}$
• laser_tilt_frame $\rightarrow$ laser_frame:

  translation: $o_{lf} = \begin{bmatrix} 0.2 & 0 & 0.1 \end{bmatrix}^T$

  position uncertainty: $\sigma^2_{p,lf} = 0$

The values chosen for the uncertainties are higher than they would normally be in reality to make the results more obvious. With these values it is possible to run the simulation. Nevertheless an additional frame for the laser point named laser_point_frame should be introduced. This artificial frame allows to calculate the uncertainty with the provided methods. It is assumed that the point is captured without uncertainty which of course does not hold in reality. However for the question that should be answered this assumption is feasible. Thus the transformation can be defined as:

• laser_frame $\rightarrow$ laser_point_frame:

  translation: $o_{lpf} = \begin{bmatrix} 2 & 0 & 0 \end{bmatrix}^T$

  position uncertainty: $\sigma^2_{p,lpf} = 0$

5.2 Discussion of the Results

To actually compare the results of the different runs, the average of the position uncertainty for one thousand transformations is used. This is an easy and sufficient enough approach to see how the tuning of parameters is influencing the quality. Figure 5.3 gives an impression where the frames are located in 3D space and how good the position quality for the laser_point_frame is at a specific point in time.

The results for the different simulations which were performed to answer the questions are listed in Table 5.1. There, only the values in millimeter of the standard deviation in $x$ and $z$-direction are shown. The variance in $y$-direction is zero and the cross-covariances can hardly be interpreted.
5.2. DISCUSSION OF THE RESULTS

<table>
<thead>
<tr>
<th>Experiment</th>
<th>laser_frame</th>
<th></th>
<th>laser_point_frame</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\sigma_x$</td>
<td>$\sigma_z$</td>
<td>$\sigma_x$</td>
<td>$\sigma_z$</td>
</tr>
<tr>
<td>10 Hz</td>
<td>4.3718</td>
<td>6.3894</td>
<td>38.370</td>
<td>84.007</td>
</tr>
<tr>
<td>10 Hz, torso joint fixed</td>
<td>4.1810</td>
<td>3.3440</td>
<td>38.214</td>
<td>82.365</td>
</tr>
<tr>
<td>10 Hz, tilt joint fixed</td>
<td>1.0488</td>
<td>5.8406</td>
<td>1.0488</td>
<td>21.349</td>
</tr>
<tr>
<td>10 Hz, both joints fixed</td>
<td>1.0488</td>
<td>2.0616</td>
<td>1.0488</td>
<td>20.597</td>
</tr>
<tr>
<td>20 Hz</td>
<td>2.3861</td>
<td>3.7239</td>
<td>19.364</td>
<td>44.207</td>
</tr>
</tbody>
</table>

Table 5.1: Standard deviation in $x$ and $z$-direction for the performed simulations.

The values show, that the uncertainty of the laser_point_frame is always a magnitude higher than the one of the laser_frame. This is as expected, because the angle uncertainty of the tilt joint has an higher impact on frames further away from the laser_tilt_frame. The tilt joint is the revolute joint which is moving the laser_tilt_frame.

The data also shows that the torso joint does not have a big influence in the position accuracy. In contrast, the tilt joint strongly influences the laser_point_frame, even if it is fixed. This is again due to the higher influence of the angle uncertainty on frames which are further away. Thus the best accuracy is achieved if the tilt joint is fixed. Last but not least the data show that if the frequency is doubled, the position uncertainty is halved. That is plausible and as expected. Nevertheless the accuracy in the simulation is limited to the case where both joints are fixed, because the frames are not moving and thus an increase of the frequency would not have any effect. The values in Table 5.1 are average values for the position uncertainty. To get the worst-case position uncertainty of a frame an additional calculation has to be performed.

The simulation and the results show that the methods presented in Chapter 4 are well suited to describe component-based systems. Further the quality measurement allows to judge transformations and enables the system integrator to configure the system in a way that the needed accuracy can be achieved.
Chapter 6

Summary and Future Work

6.1 Summary

Service robots with their numerous actuators and sensors on board, where each defines an own coordinate frame, need a way to easily do coordinate transformations. Therefore, robotic frameworks have to provide methods to efficiently distribute TF Data between components and to provide a quality measurement for a transformation. This quality measurement enables the system integrator to parametrize the system to get the needed accuracy for a specific application.

In this work, an abstract representation of component-based robotic systems and the data flow between components which is related to transformations was introduced. This representation allows to judge the design of existing systems and also helps to design new systems. Further, together with the tf tree, it is possible to see which component is responsible for which transformation. The examples and the simulation in Chapter 5 showed that the proposed representation is suited to describe any system in relation to transformations.

The quality measurement based on a 3D Gaussian normal distribution allows to describe the position uncertainty at a point in time of a frame relative to another frame. It was shown that it is possible to calculate the quality of one transformation but also of a transformation chain. Further, the measurement allows the system integrator to see which transformation in a chain introduces the most uncertainty. With this knowledge, counter-measures like the one presented in Section 4.4.4 can be taken to increase the quality.

Finally, the values for the Communication Object which contains the TF Data for one transformation was presented. The CommObject ensures a clear separation of components and thus allows reusability of them in different applications.

This work has introduced an abstract representation of component-based systems in relation to transformations. Further, it showed the benefit of using a quality measurement for transformations which was not used in robotics until now. It puts the system integrator in a position to better understand the system and to configure it in a way to reach the needed accuracy for an application.

6.2 Future Work

This thesis provides the fundamentals for managing coordinate frame transformations in a distributed, component-based robot system. It showed the value of an abstract system representation and a quality measurement for transformations.
One next step is to extend frameworks like ROS or SMARTSOFT which already come with some built in support for tf. This can be done by using the methods presented in Chapter 4 and should start with the analysis of the existing systems by using their abstract representations. Further the simulation used in Chapter 5 can be used to simulate parts of existing systems to determine the quality of transformations without modifying the existing system. To actually do this the simulation framework must be extended to allow better analysis and easier modelling of different systems. As an example how a simulation framework can be designed is DESMO-J [11], which is a framework for Discrete-Event Modelling and Simulation written in Java. The framework provides a lot of methods for modeling event based systems, but it has a lack of tools to visualize 3D data. That is why the framework was not chosen at the first time in this thesis.

In general tools must be provided to support the system integrator with the configuration of the system. For these tools to work, component and hardware developers have to provide information about the uncertainties of actuators like position and velocity uncertainty. Further an integration of these tools in existing toolchains like the SMARTSOFT MDSD Toolchain [30] is desirable to use existing information and to provide a consistent view on the system.

Until now only tf frames which describe the robot are considered. Further investigation should be taken to extend the proposed methods to object frames or the robot frame in relation to the world frame. For example, this extend would allow the system integrator to see if the position of an object is known accurate enough to grasp it. The calculations should be mostly the same as the ones presented in this work.
# List of Figures

1.1 Three state-of-the-art robots. .................................................. 1
1.2 Components which allow to do collision avoidance. ....................... 2
1.3 Abstract robot model consisting of links and joints. .................... 3
1.4 General tf tree. ..................................................................... 3

3.1 Relationship between frames and joints. ................................. 9
3.2 Translation of a square in $\mathbb{R}^2$. ...................................... 10
3.3 Rotation of a square in $\mathbb{R}^2$. .......................................... 11
3.4 Quaternion Operations on Vectors. [17, p. 117] .......................... 14
3.5 Relationship between clocks when they tick at different rates. ....... 15
3.6 Way to get the current time from a time server. .......................... 16
3.7 Clock synchronization in PTP. .................................................. 17

4.1 Components needed for collision avoidance. ............................. 20
4.2 Hardware setup and tf tree of a statically mounted laser scanner. .... 20
4.3 Hardware setup and tf tree of a tilting laser scanner. .................. 21
4.4 Tilting laser scanner moves the laser scan from the xy-plane to an oblique plane. 21
4.5 Hardware setup and tf tree of a sensor mounted on a robot arm. ....... 22
4.6 Components needed to capture a 3D point cloud. ....................... 22
4.7 The elements to represent different systems. .............................. 26
4.8 How the different elements can be combined to form a system. ......... 27
4.9 TF Data labeled with names that link the data to the frame in the tf tree. 28
4.10 Mapping of polar to Cartesian coordinates. .............................. 30
4.11 Uncertainty in polar and Cartesian coordinates. ......................... 31
4.12 Tf tree with three frames. .................................................... 32
4.13 Single uncertainties for the frames. ........................................ 33
4.14 Complete tf chain with the single uncertainties and the total uncertainty. 34
4.15 Different types of matching time stamps. .................................. 36
4.16 Gaps and overlaps in range of valid data. ............................... 36
4.17 Valid TF Data with overlaps but without gaps. ......................... 37
4.18 Switch from fixed state to moving state. .................................. 37
4.19 Simple tf tree with two frames. ............................................. 38
4.20 Tf tree for the example setup. .............................................. 39
4.21 Modelling of the ROS tf system. .......................................... 40
4.22 Modelling of the current tf support in SMARTSOFT. .................. 40
4.23 Model where a central component manages all actuators. ............. 41
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.24</td>
<td>Mean and covariance for $t_0$ and $t_1$</td>
<td>43</td>
</tr>
<tr>
<td>4.25</td>
<td>$Tf$ tree and frames for a transformation chain.</td>
<td>43</td>
</tr>
<tr>
<td>4.26</td>
<td>Time where $TF$ Data was captured.</td>
<td>44</td>
</tr>
<tr>
<td>4.27</td>
<td>$Tf$ chain with $3\sigma$ uncertainty ellipses.</td>
<td>47</td>
</tr>
<tr>
<td>5.1</td>
<td>$Tf$ tree of a robot with different actuators and sensors.</td>
<td>49</td>
</tr>
<tr>
<td>5.2</td>
<td>Component model with data flow. $TF$ Data is linked by names with the $tf$ tree.</td>
<td>50</td>
</tr>
<tr>
<td>5.3</td>
<td>Frames in 3D space with $3\sigma$ uncertainty ellipsoid for the激光点frame.</td>
<td>52</td>
</tr>
</tbody>
</table>
List of Tables

3.1  Comparison of matrices, Euler angles, and quaternions. [8, p. 179] . . . . . . . . . . 14
5.1  Standard deviation in $x$ and $z$-direction for the performed simulations. . . . . . . . 53
Bibliography


