Christoph Hubert Klug

# Assistance for Measuring Human-Made Structures with Robotic Total Stations

**DOCTORAL THESIS**

to achieve the university degree of
Doktor der technischen Wissenschaften

submitted to

**Graz University of Technology**

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Dieter Schmalstieg
Institute for Computer Graphics and Vision

Advisor

Dipl.-Ing. Dr.techn. Clemens Arth
Institute for Computer Graphics and Vision

Graz, Austria, March 2019

To my beloved wife Karina.

I think the thing to do is enjoy the ride while you're on it.

*Johnny Depp*

# Abstract

The history of total stations dates back to the 1970s, when manufacturers combined a theodolite with a laser distance meter for measuring angles and distances with high accuracy. The integration of multiple sensors over the years turned the system into a smart and powerful measurement device. Modern total stations consist of a variety of different sensors and actuators, a tracking system for reflective prisms, an embedded processor and a remote control unit to assist users with standard measuring tasks. Such systems allow assisted targeting and tracking, and apply automatic measurement corrections when using surveying prisms.

However, with all the sensors in place, traditional methods do not use the full potential of totals stations, especially when using the reflectorless measurement mode. In particular, the quality of measuring natural targets highly depends on the user experience.

In this work, we address measurement assistance systems for reflectorless robotic total stations in the field of surveying and building construction. To target a wide range of devices, we focus on systems that do not have explicit sensor data synchronization and do not rely on photogrammetry. Our methods increase the productivity and allows non-experts to perform accurate and reliable measurements.

In particular, we present an assistance system for accurate targeting of human-made structures with a robotic total station in reflectorless mode. We reduce the uncertainty and increase the reliability of corner and edge measurements by applying linear approximations of the measured structure in real-time.

Furthermore, we present an assisted reflectorless registration of a robotic total station and a CAD model. Here, we reduce the required user interaction, while retaining accurate and reliable registration in real-time.

In this work, we use a generalized description of robotic total stations based on robotic theory. We present all required steps for converting the system model into an efficient design and simulation ecosystem. This allows exploration of the problem and solution space beyond the limitations of particular hardware configurations, and seamless exchange of the simulator and physical devices for prototyping and concept verification. In particular, we discuss how geometric models of robotic total stations can be extracted automatically by using the Denavit-Hartenberg convention. We discuss modeling concepts for various sensor and environment combinations and analyze the simulation uncertainty of an exemplary setup. In addition, we qualify the simulator according to the *JCGM 100:2008 Guide to the Expression of Uncertainty* (GUM), which describes the evaluation and report of physical measurements and their measurement uncertainties. This allows for a standardized comparison of similar systems and interpretation of simulation results. We also present an a-priori qualification method, which allows specifying crucial parameters for simulation setups in advance to the actual implementation. The method is intended to serve researchers, software and hardware developers as a guide for designing simulation and verification systems with similar properties.

# Kurzfassung

Totalstationen wurden schon in den 1970ern im Bereich der Geodäsie eingesetzt, um Horizontalrichtungen, Vertikalrichtungen und Distanzen gleichzeitig zu bestimmen. Diese Geräte basieren auf der Kombination von Theodoliten mit elektrooptischen Entfernungsmessern und ermöglichen dadurch hochgenaue Messungen auf große Entfernungen. Mittels Stellmotoren und zusätzlich verbauten Sensoren werden bei modernen Geräten nicht nur Umwelteinflüsse kompensiert und Messfehler verringert, sondern auch Arbeitsabläufe vereinfacht und automatisiert. Beispielsweise gehören automatisches Messen und Tracking von retroreflektierenden Messprismen, reflektorloses Messen von diffus reflektierenden Oberflächen, bildgestütztes Anzielen von Objekten, Funkfernsteuereinheiten sowie die Speicherung und Verarbeitung von Messdaten durch integrierte Prozessoreinheiten mittlerweile zur Standardausführung.

Trotz der Möglichkeiten, die diese Sensoren und Aktoren bieten, greifen viele Arbeitsabläufe noch auf traditionelle und eher konservative Ansätze zurück. Speziell bei der reflektorlosen Betriebsart sind Anwendererfahrung und entsprechende Schulungen entscheidend für die Qualität und Wiederholbarkeit von Messungen. So sind zum Beispiel Gebäudeecken wegen ihres hohen Wiedererkennungswertes beliebte Messziele, verursachen aber durch Kantenabrundungen Messunsicherheiten. Zusätzlich können Unterschiede zwischen verschiedenen Anwendern beim manuellen Anzielen solcher Strukturen kaum vermieden werden.

In dieser Dissertation stellen wir zwei Assistenzsysteme für ausgewählte Arbeitsabläufe für reflektorloses Messen mit Totalstationen vor. Unsere Methoden erhöhen die Produktivität von Arbeitsabläufen und erleichtern speziell Laienanwendern die Durchführung der ausgewählten Aufgaben. Dabei reduzieren wir die Geräteanforderungen auf wesentliche Funktionalitäten, um die Portierbarkeit und praktische Umsetzbarkeit zu gewährleisten. Deshalb vermeiden wir photogrammetrische Bildverarbeitung und arbeiten mit nicht synchronisierten Messdaten.

Zum einen betrachten wir ein Assistenzsystem für das reflektorlose Messen von Strukturen auf Baustellen. Mittels Labortests und einer Pilotstudie zeigen wir, dass lineare und lokale Approximationen von Gebäudeecken in bekannte Messabläufe einfach integriert und die Messunsicherheit und Messwiederholbarkeit dadurch deutlich verbessert werden können.

Zum anderen beschreiben wir ein Assistenzsystem, um die reflektorlose Verortung einer Totalstation mit Hilfe eines CAD-Modells im Innenbereich zu vereinfachen. Unser System reduziert die benötigte Benutzereingabe auf die Auswahl und das ungefähre Anzielen einer Ecke, und entkoppelt damit die Messunsicherheit der Verortung von der manuellen Zielgenauigkeit.

Des Weiteren leiten wir im Zuge dieser Arbeit eine allgemeine Beschreibung der Positionskinematik von Totalstationen mit Hilfe von bekannten Methoden aus der Robotik her. Dazu stellen wir alle notwendigen Schritte vor, um das kinematische Modell in ein Simulations- und Entwicklungssystem für interaktive Algorithmen überzuführen. Das bietet die Möglichkeit, effizient und explorativ Arbeitsabläufe zu entwickeln und auch über aktuelle Hardwarelimitierungen hinauszublicken.

Im Detail beschreiben wir, wie Denavit-Hartenberg-Parameter anhand von Gerätetreiber und Standardfunktionen dieser Instrumente extrahiert werden können, ohne dafür externe Messaufbauten zu benötigen. Hierbei untersuchen wir Simulationsmodelle für verschiedene Sensoren und verschiedene Messszenarien hinsichtlich der zu erwartenden Simulationsunsicherheiten. Neben analytischen Verfahren und Monte-Carlo-Simulationen diskutieren wir hierzu auch A-Priori-Abschätzungsmethoden. Zur Verbesserung der Vergleichsmöglichkeiten unserer Methoden und Ergebnisse mit vergleichbaren Ansätzen stellen wir einen exemplarischen Simulator und dessen Messunsicherheiten entsprechend des Leitfadens *JCGM 100:2008 Guide to the Expression of Uncertainty* (GUM) vor, welcher die Abschätzung und Angabe der Unsicherheit beim Messen standardisiert.

## Affidavit

*I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material, which has been quoted either literally or by content from the sources used.*

*The text document uploaded to TUGRAZonline is identical to the present doctoral thesis.*

________________________ ________________________

Date Signature

# Acknowledgments

First and foremost I would like to thank my wife Karina for her unbelievable support. She has been and always will be my love and inspiration.

Special thanks to my team leader and mentor Dr. Clemens Arth, and to my supervisor Prof. Schmalstieg, who always provided the right amount of guidance and are always inspiring new ideas.

I am very thankful to my family for always supporting me in reaching my goals. Especially my brother Ferdinand, my sister-in-law Carmen, and my niece Pia-Kristin showed incredible patience and generosity in sharing their flat with me when my wife and I moved to a different area in Austria at the end of my studies. Ferdinand also continuously provided me with practical advice for my ideas and offered his expertise throughout the complete project.

I would like to thank our industrial partner for the financial support, the successful and effective cooperation, and the opportunity for regularly and continuous discussions. In particular, Thomas Gloor, our team leader on the industrial side, was a source of inspirations, ideas and feedback. The time, effort and constant support of Thomas made it even possible at all to bridge the gap between research and industry.

In addition, I also would like to thank my colleagues at the VRVis and the ICG for the opportunity to work in a dynamic and inventive team on leading-edge technologies for the industry and my PhD. Our IT and office staff are simple wonderful, and the lively and critical discussions with our research teams were always constructive and certainly increased the quality of my work.

This goes out to my family, friends, colleagues and supporters: It was a joyful and wild journey, and it was worth every second! Thank you so much!

# Contents

# 1 Introduction

Total stations are electronic measurement devices, which are used in surveying and construction [106].

The history of total stations dates back to the 1970s when manufacturers combined a theodolite with a laser distance meter for measuring angles and distances simultaneously with high precision and accuracy.

A theodolite consists of a movable telescope, which is mounted on a fixed base. The telescope can rotate horizontally and vertically and provides angular readouts. These angles can be used to relate multiple measured targets. Electronic theodolites contain rotary encoders or inclinometers for reading out angular values automatically. Traditional devices provide an optical telescope and allow manual targeting of interest points (IP) in high distances. A crosshair in the optical eyepiece indicates the sighting axis. For convenience of operation, the instrument can be mounted on a tripod.

Appearance and operating principle of total stations are similar to modern theodolites, except for the additional electronic distance meter (EDM), which is built into the telescope housing. The laser ray is aligned with the sighting axis of the theodolite and is again indicated by the crosshair in the optical eyepiece. Compared to theodolites, which are designed to measure angles only, total stations can additionally measure distances and 3D points. Sensor data fusion, conversion between different measurement representations, and estimation of control parameters for targeting a particular IP are done internally by firmware or device drivers.

Recent devices do not have an optical eyepiece, but one or more cameras to allow remote device controlling. Modern instruments include additional components, such as temperature and gravity sensors, a compass, an automatic target tracking unit, a GPS module, servos, an embedded computer and a graphical user interface (UI). These devices are referred to as image-assisted robotic total station, or short RTS, in this work.

A simple geometric model for an RTS is a spherical coordinate system with no parallax effects between the coordinate systems of sensors and actuators. Adjustment screws are

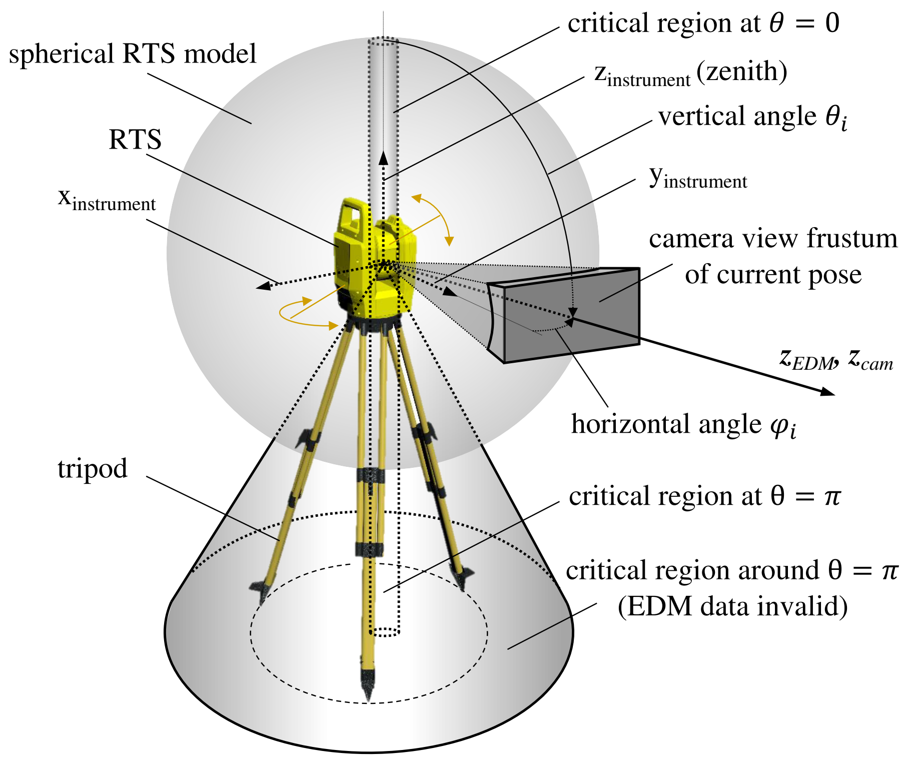

spherical RTS model
RTS
$x_{\text{instrument}}$
critical region at $\theta = 0$
$z_{\text{instrument}}$ (zenith)
vertical angle $\theta_i$
$y_{\text{instrument}}$
camera view frustum of current pose
$z_{EDM}$, $z_{cam}$
horizontal angle $\varphi_i$
tripod
critical region at $\theta = \pi$
critical region around $\theta = \pi$ (EDM data invalid)

Figure 1.1: Actuators, sensors and special regions of an exemplary RTS.

used to approximate the ideal model during hardware calibration. Detailed models can be used to allow for better calibration between the individual components and to include camera parameters, sensor nonlinearities, and temperature influences. High-end devices additionally compensate various system effects automatically, such as inaccuracies in production, sensor drifts and environmental influences [106]. Figure 1.1 shows a modern RTS with one camera and the superimposed, parallax-free geometric model. Details about RTS, device components, calibration and models can be found in the book written by Uren and Price [106] and in the work of Amann et al. [3].

A comparison of modern RTS and selected properties is provided in Table 1.1. Virtually all RTS nowadays support at least two targeting modes, namely measuring with and without reflectors.

In the former case, the measurement object is a *reflective target*. Reflective targets are retro-reflective prisms or tapes that are specially designed to lower the measurement error. 3D measurements up to a few kilometers are possible, and automatic targeting and target tracking can be used to simplify certain workflows [106].

The *reflectorless mode* allows measuring *natural targets* with diffuse surface reflection directly and therefore does not require physical installations or additional equipment.

However, the distance range in this situation is significantly lower, and higher measurement uncertainties are to be expected. Typical update rates for 3D measurements in this mode ranges from 0.5 to 50 Hz, not counting the required time to steer the device between different directions [31].

Modern devices have a variety of applications. General use cases for RTS include measuring and registering natural and artificial features in 2D and 3D maps. Examples for applications are: topographic surveying, setting out features like pipes, land and house boundaries, roads and tunnels, and as-built surveying. Other applications include recordings and reconstructions on archaeological sites, crime scenes and road traffic accidents, and quality control on sites.

Despite the increasing technological progress in computer vision and robotics in recent decades, many RTS tasks still require high user experience for robust, reliable and repeatable measurements. Figure 1.2 shows common measurement configurations and targeting challenges. Ehrhart [31] provides a recent treatise on applications and calibration methods for image-assisted RTS. The author emphasizes that many traditional workflows use the RTS image data mainly for targeting and documentation, and highlights the potential benefits of image-assisted algorithms for various scenarios.

## 1.1 Contribution

A common task in the building construction is the *reflectorless measurement* of *natural targets*. Corners and edges of human-made structures are preferred natural targets in building construction because of their high recall value. While most modern devices include assistance systems for measuring reflective targets, natural targets have a huge variety and therefore are still measured manually [31]. Image-based targeting systems are limited by camera resolution, environmental influences, model and calibration errors, shadow effects and low-contrast targets. In addition, laser beam divergence, angular resolution of the theodolite, and targeting uncertainty increase the measurement uncertainty and reduce the reliability of non-planar targets. These effects render fully automated approaches critical for surveying.

In this work, we present a vision-based assistance system for measuring Manhattan-like structures. The system combines user input with linear approximations of the target, provides visual feedback and user-control in real-time. In particular, we test a variety of linear 3D approximations, which we call support objects. We explicitly avoid photogrammetric approaches to cover a broad range of systems and to include parallax-free systems without additional overview cameras. We show that our method can reduce the uncertainty of such measurements, without compromising the task efficiency.

Another common task in building construction is the registration of RTS measurements

Table 1.1: Common properties of modern RTS as published by Lachat et al. [70], but extended by recent price values and the modern device *Trimble RPT600*. (Courtesy Topcon Positioning Systems Inc., Leica Geosystems Inc., Trimble Inc.)

| property | Topcon IS-3 | Leica Nova MS60 | Trimble SX10 | Trimble RPT600 |
|---|---|---|---|---|
| device image | | | | |
| **Release Date and Price** | | | | |
| release date | 2011 | 2015 | 2016 | 2018 |
| approximate price (2018) | $15\times10^3$ Euro | $45\times10^3$ Euro | $35\times10^3 - 65\times10^3$ Euro | $20\times10^3$ Euro |
| **Angle and Distance Measurement** | | | | |
| angular accuracy | $1-5''$ | $1''$ | $1''$ | $1''$ |
| EDM maximum range (prism) | $5\times10^3$m | $10\times10^3$m | $5.5\times10^3$m | 100m |
| EDM maximum range (reflectorless) | $2\times10^3$m | $2\times10^3$m | $0.8\times10^3$m | |
| tracking range | - | $1\times10^3$m | 800m | 100m |
| EDM distance accuracy (prism) | 2mm + 2ppm | 1mm + 1.5ppm | 1mm + 1.5ppm (ISO 17123-4) | 3mm @ 50m + 10ppm (ISO 17123-5) |
| EDM distance accuracy (reflectorless) | 3mm MSE (fine mode up to 250m) 10mm + 10ppm MSE (long mode up to 2 exp 3m) | 2mm + 1.5ppm (ISO 17123-3) | 2mm + 1.5ppm (ISO 17123-4) | |
| measurement time (prism, std.) | - | 1.5s | 1.6s | 2.5s |
| measurement time (reflectorless, std.) | - | 1.5s | 1.2s | 3 – 15s |
| measurement time (prism, tracking) | - | - | - | 0.4s |
| measurement time (reflectorless tracking) | - | - | - | 0.4s |
| **Imaging** | | | | |
| number and type of cameras | wide-angle + coaxial | overview + telescope | overview + primary + coaxial | telescope |
| resolution | $1.3\times10^6$ pixel | $5\times10^6$ pixel | $5\times10^6$ pixel | 0.36 exp 6 pixel |
| frame rate | up to 10 Hz | up to 20 Hz | up to 15 Hz | - |
| **Scanning** | | | | |
| maximum rate | 20 pts/s | 1 exp 3 pts/s @ 300m | 26.6 exp 3 pts/s | - |
| maximum range | $2\times10^3$m | $2\times10^3$m @ 1 Hz | $0.6\times10^3$m | - |
| scanning range noise | - | 1mm @ 50m | 1.5mm @ 50m | - |

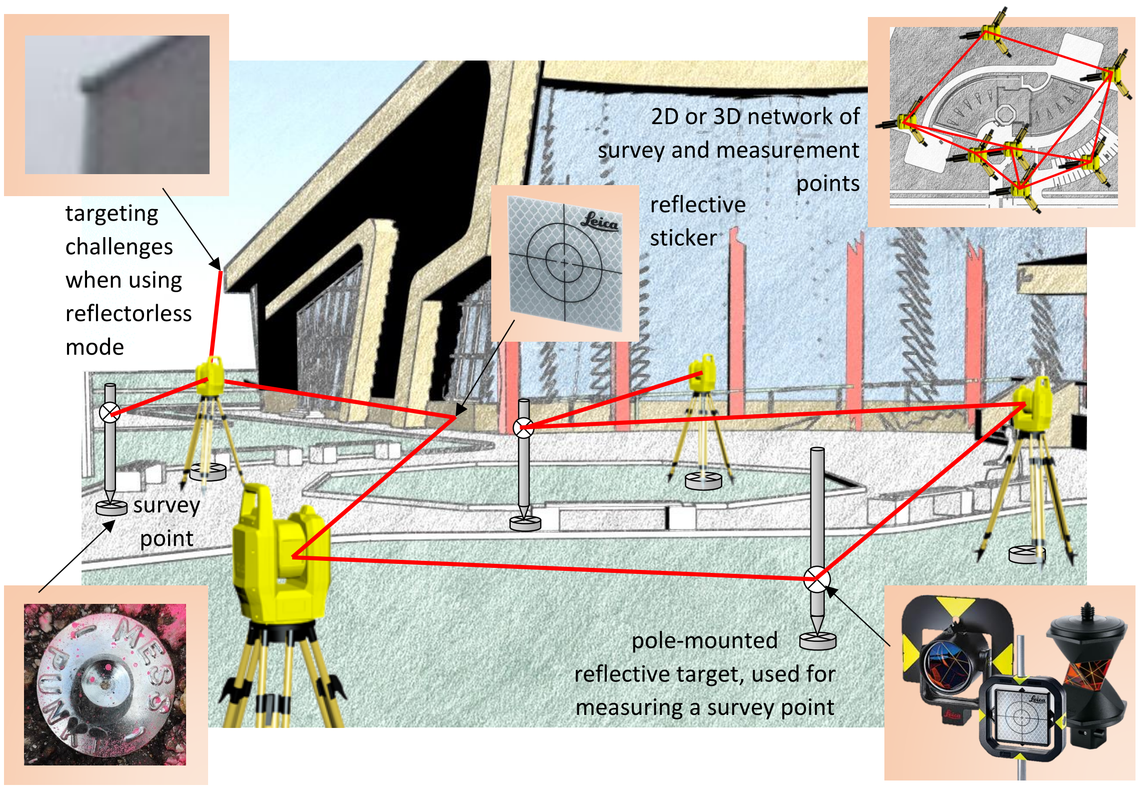

2D or 3D network of survey and measurement points
reflective sticker
targeting challenges when using reflectorless mode
survey point
pole-mounted reflective target, used for measuring a survey point

Figure 1.2: Typical RTS measurement setup for 2D and 3D mapping. Reflectors can be mounted on a measurement pole or fixed to a surface (courtesy Leica Geosystems AG.). Triangulation, trilateration, triangulateration and traversing are used to register measurements by using previously measured points [106].

with respect to a CAD model. Here, fast RTS registration without dedicated control points is desired. Such registration is used for planning and documenting installations, for placing physical markers for assembly drills, and for planning reconstructions.

We present an assistance system for the registration of sparse RTS measurements with respect to a polygonal CAD model. Similar to the workflow proposed by Bosché [16, 17], we split the procedure into coarse and fine registration. The modular approach allows adaption of the system to different measurement tasks, such as varying the coarse registration method or skipping the fine registration step. With our workflow, coarse registration remains mainly user-driven. However, compared to traditional approaches, our method reduces the required user interaction and the accuracy demands on the manual measurements from accurate targeting multiple points to defining the current visible area in the CAD model. The coarse registration is used for selecting potential visible areas in the CAD model and for automatic sampling additional points to refine the registration. The system further supports visual overlay of the CAD model to allow for visual inspection of the registration on the construction side and to increase the registration reliability. We show that our method is in the same accuracy range as traditional methods, but has significantly lower user-constrains for indoor-registration. In particular, we show that the Iterative Closest Point (ICP) algorithm can be beneficial in certain scenarios even with the sparsity of RTS measurement sets.

The workflows mentioned above are of user-centeric and interactive nature, which optional overlay of visual content for verification reasons. While RTS are powerful devices with a long history in surveying and construction, many problems arise when exploring user-centric, interactive or tele-operated schemes.

To begin with, RTS were not designed for computer vision or augmented reality applications, and special research requirements on driver and firmware were not met by RTS, as stated by Ehrhart [31].

Likewise, synchronized data, low latency, and hardware-in-the-loop (HIL) approaches for algorithm design and verification are core requirements for such applications, as described in the book by Schmalstieg and Höllerer [92]. However, as the traditional RTS tasks for non-reflective targets built upon measuring individual targets manually, the focus is on power efficiency, error-free operation and high accuracy rather than on low latency of the data stream. Conversion of the raw data to 3D measurements is done internally by the device, and therefore no synchronization data is required on the application layer.

Similar situations in robotic research arises when exploring prototypical hardware, working on the frontier of the latest technological developments or experimenting with leading-edge devices: Properties, drivers and documentations are often work-in-progress, and details are company secrets during early design phases.

For example, accurate Augmented Reality (AR) systems rely on low discrepancies between hardware and model, but also require low latency to maintain visual coherency and reasonable reactivity for user interactions; following the *principle of locality*, measurement and model data conversion should be handled locally; continuous hardware access increases the latency and therefore should be avoided if possible; dynamic movement predictions require a proper geometric device model and at least time stamps for all sensor data. However, with insufficient access to raw sensor data, no possibility to modify firmware and drivers, wireless communication channels, and without detailed insight into the geometric models and the applied corrections, the possibilities for local motion prediction and data conversion on the application layer are limited.

Finally, environmental effects can often not be controlled or modeled by individual error sources; the installation of physical targets is time consuming, the exploration of device variations not possible without adjustments of the physical device.

As a result, maintaining a suitable hardware and software ecosystem for RTS research can involve a disproportionate effort, or, as worst-case scenario, is not possible at all.

In this work, we use hardware simulation with appropriate device and environmental models to overcome many of the issues mentioned above.

We provide a complete workflow to extract the geometric model parameters using the

Denavit-Hartenberg (DH) convention [28]. In particular, we extend the modeling methods of Barker [10] and Rajeevlochana et al. [84] for RTS. We further present a system to analyze the discrepancy between the extracted models and the models used by hardware drivers.

While extracted DH parameters can be used with existing robotic simulators, we present the required steps for converting the extracted geometric model to a custom RTS prototyping and simulation framework. Here, the simulation uncertainty is a crucial information for designing simulation setups and for interpreting simulation results for RTS measurement setups. We provide methods to analyze the simulation uncertainty analytically and with Monte Carlo experiments. While standard Monte Carlo experiments are limited to a-posteriori qualification of a particular configuration, we present a generalized method for a-priori qualification of RTS simulators and of systems with similar hardware configurations.

To summarize, the main objective of this work is to enhance RTS jobs by means of novel user-assistance systems for two selected scenarios.

A secondary objective is to provide a complete workflow for generating efficient ecosystems for developing and testing interactive workflows, even beyond the actual hardware capabilities of a particular instrument.

Figure 1.3 shows the taxonomy of the objectives, contributions and the related chapters of this work.

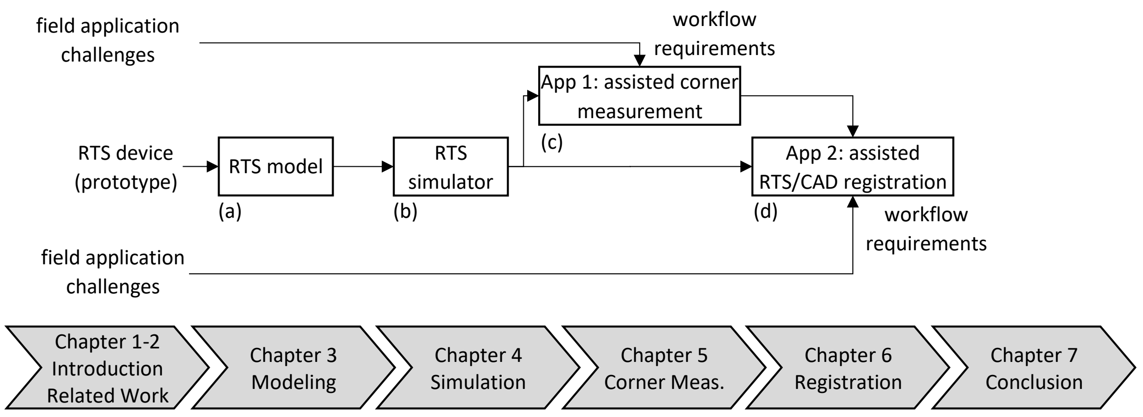


Figure 1.3: Taxonomy of objectives, contribution and document structure of this thesis. The main goal of this work is to ease corner measurement and device registration by novel assistance systems without specific image analysis. RTS modeling and simulation increases the efficiency of designing and testing of interactive algorithms and when working with RTS prototypes. Our contributions are in the field of: (a) device modeling [66], (b) realtime simulation [63], (c) assisted corner measuring [67], and (d) RTS-CAD registration [64].

## 1.2 Collaboration Statement

This work was enabled by Graz University of Technology and by the Competence Center VRVis. VRVis is funded by a grant from the Competence Centers for Excellent Technologies (COMET) 843272.

### 1.2.1 List of Publications

My work in this project at the Institute for Computer Graphics and Vision (TU Graz) and at the Competence Center for Virtual Reality and Visualization (VRVis Vienna) led to the five peer-reviewed publications, which are listed below in chronological order[1].

| | | |
|---|---|---|
| Title: | **Measuring Human-made Corner Structures with a Robotic Total Station using Support Points, Lines and Planes** [65] | |
| Authors: | Klug, Christoph and Schmalstieg, Dieter and Arth, Clemens | |
| In: | *Proceedings of the International Conference on Computer Vision Theory and Applications (VISAPP)* | |
| Published: | February 2017 | |
| Contributions: | Klug, Christoph | first author, major contributor in terms of concept, design, implementation, test and publication of the work |
| | Schmalstieg, Dieter | supervisor; revising and final approval of the publication |
| | Arth, Clemens | advisor; managing academic aspects of the project, revising and final approval of the publication |
| | Gloor, Thomas | contributor; managing industrial aspects of the project |
| Abstract: | Measuring non-planar targets with a total station in reflectorless mode is a challenging and error-prone task. Any accurate 3D point measurement requires a fully reflected laser beam of the electronic distance meter and proper orientation of the pan-tilt unit. Prominent structures like corners and edges often cannot fulfill these requirements and cannot be measured reliably. We present three algorithms and user interfaces for simple and efficient construction-side measurement corrections of the systematic error, using additional measurements close to the non-measurable target. Post-processing of single-point measurements is not required with our methods, and our experiments prove that using a 3D point, a 3D line or a 3D plane support can lower the systematic error by almost a order of magnitude. | |

[1]Some passages of these publications are included verbatim in this work.

| | | |
|---|---|---|
| Title: | **A Complete Workflow for Automatic Forward Kinematic Model Extraction of Robotic Total Stations Using the Denavit-Hartenberg Convention** [66] | |
| Authors: | Klug, Christoph and and Schmalstieg, Dieter and Gloor, Thomas and Arth, Clemens | |
| In: | *Journal of Intelligent and Robotic Systems (JIRS)* | |
| Published: | September 2018 | |
| Contributions: | Klug, Christoph | first author, major contributor in terms of concept, design, implementation, test and publication of the work |
| | Schmalstieg, Dieter | supervisor; revising and final approval of the publication |
| | Gloor, Thomas | managing industrial aspects of the project, revising and final approval of the publication |
| | Arth, Clemens | advisor; managing academic aspects of the project, revising and final approval of the publication |

Abstract:

Development and verification of real-time algorithms for robotic total stations usually require hard- ware-in-the-loop approaches, which can be complex and time-consuming. Simulator-in-the-loop can be used instead, but the design of a simulation environment and sufficient detailed modeling of the hardware are required. Typically, device specification and calibration data are provided by the device manufacturers and are used by the device drivers. However, geometric models of robotic total stations cannot be used directly with existing robotic simulators. Model details are often treated as company secrets, and no source code of device drivers is available to the public. In this paper, we present a complete workflow for automatic geometric model extraction of robotic total stations using the Denavit-Hartenberg convention. We provide a complete set of Denavit-Hartenberg parameters for an exemplary robotic total station. These parameters can be used in existing robotic simulators without modifications. Furthermore, we analyze the difference between the ex- tracted geometric model, the calibrated model, which is used by the device drivers, and the standard spherical representation for 3D point measurements of the device.

| | | |
|---|---|---|
| Title: | **Measurement Uncertainty Analysis of a Robotic Total Station Simulation** [63] | |
| Authors: | Klug, Christoph and Arth, Clemens and Schmalstieg, Dieter and Gloor, Thomas | |
| In: | *Proceedings of the IEEE International Conference on Industrial Electronics, Control, and Instrumentation (IECON)* | |
| Published: | October 2018 | |
| Contributions: | Klug, Christoph | first author, major contributor in terms of concept, design, implementation, test and publication of the work |
| | Arth, Clemens | advisor; managing academic aspects of the project, revising and final approval of the publication |
| | Schmalstieg, Dieter | supervisor; revising and final approval of the publication |
| | Gloor, Thomas | managing industrial aspects of the project, revising and final approval of the publication |

Abstract: The design of interactive algorithms for robotic total stations often requires hardware-in-the-loop setups during software development and verification. The use of real-time simulation setups can reduce the development and test effort significantly. However, the analysis of the simulation uncertainty is crucial for proper design of simulation setups and for the interpretation of simulation results. In this paper, we present a real-time simulation method for modern robotic total stations. We provide details for an exemplary robotic total station including models of geometry, actuators and sensors. The simulation uncertainty was estimated analytically and verified by Monte Carlo experiments.

| | | |
|---|---|---|
| Title: | **Semi-Automatic Registration of a Robotic Total Station and a CAD Model Without Control Points** [64] | |
| Authors: | Klug, Christoph and Arth, Clemens and Schmalstieg, Dieter and Gloor, Thomas | |
| In: | *Proceedings of the IEEE International Conference on Industrial Electronics, Control, and Instrumentation (IECON)* | |
| Published: | October 2018 | |
| Contributions: | Klug, Christoph | first author, major contributor in terms of concept, design, implementation, test and publication of the work |
| | Arth, Clemens | advisor; managing academic aspects of the project, revising and final approval of the publication |
| | Schmalstieg, Dieter | supervisor; revising and final approval of the publication |
| | Gloor, Thomas | managing industrial aspects of the project, revising and final approval of the publication |

Abstract: The accurate registration of a robotic total station with respect to a given CAD model is a crucial task in the construction industry. Common registration techniques rely on a reference network of control points in the CAD model. One must establish correspondences between control points in the CAD model and measured points in the field. Usually physical markers or natural points of interest are selected as control points. We present a user-guided algorithm for simple and efficient registration of a robotic total station with a CAD model in indoor environments without the need for control points. The user interaction is reduced to selecting a local Manhattan-like corner structure for initial model alignment; accurate registration of the device is carried out automatically. Our algorithm relies on angle and distance measurements only and, therefore, is not limited to vision-based robotic total stations. In particular, we propose a new algorithm for robust Manhattan corner extraction.

| | | |
|---|---|---|
| Title: | **On Using 3D Support Geometries for Measuring Human-made Corner Structures With a Robotic Total Station** [67] | |
| Authors: | Klug, Christoph and Schmalstieg, Dieter and Gloor, Thomas and Arth, Clemens | |
| In: | *Computer Vision, Imaging and Computer Graphics – Theory and Applications (Revised Selected Papers of VISIGRAPP 2017)* | |
| Published: | January 2019 | |
| Contributions: | Klug, Christoph | first author, major contributor in terms of concept, design, implementation, test and publication of the work |
| | Schmalstieg, Dieter | supervisor; revising and final approval of the publication |
| | Gloor, Thomas | managing industrial aspects of the project, revising and final approval of the publication |
| | Arth, Clemens | advisor; managing academic aspects of the project, revising and final approval of the publication |

Abstract: Performing accurate measurements on non-planar targets using a robotic total station in reflectorless mode is prone to errors. Besides requiring a fully reflected laser beam of the electronic distance meter, a proper orientation of the pan-tilt unit is required for each individual accurate 3D point measurement. Dominant physical 3D structures like corners and edges often do not fulfill these requirements and are not directly measurable. In this work, three algorithms and user interfaces are evaluated through simulation and physical measurements for simple and efficient construction-side measurement correction of systematic errors. We incorporate additional measurements close to the non-measurable target, and our approach does not require any post-processing of single-point measurements. Our experimental results prove that the systematic error can be lowered by almost an order of magnitude by using support geometries, i.e. incorporating a 3D point, a 3D line or a 3D plane as additional measurements.

# 2 Related Work and State of the Art

Without the claim of completeness, this section provides literature about RTS fundamentals. We review published work about RTS that is related to device modeling and simulation, reflectorless measurements of natural targets and RTS assistance algorithms for device registration. In particular, the focus is on Manhattan-like corner measurements and device registration with respect to a CAD model.

## 2.1 Traditional Surveying

Traditional surveying methods are described in Uren and Price [106], Coaker [23] and Zeiske [111]. The authors provide introductions to surveying devices and methods, including standardized mathematical models, measurement methods, common workflows and error estimations. They also provide extensive descriptions of the individual components of total stations, such as GPS modules, laser range meters, and the electronic theodolite. In addition, basic surveying and measurement methods, surveying hardware, software and tools, and possible sources of errors are handled. A common RTS measurement setup is shown in Figure 1.2.

Modern total stations support image-based measurement methods, such as steering the RTS to selected pixels, selecting and visualizing 3D targets in the image or visualizing metadata. For instance, the Topcon device [25] includes image-assisted targeting and measurement features for corners and edges, but without giving any scientific details or uncertainty assessments of the features. Coaker [23] explores quality and reliability properties of reflectorless RTS measurement methods, and also discusses problems with measuring corners and edges directly.

The references mentioned above provide fundamentals of reflectorless measuring. However, the authors do not address measurement assistance or simulation and verification systems for interactive workflow.

In the following sections, we provide published work related to the individual topics.

## 2.2 Modeling of Robots

Kinematic modeling and model identification has been addressed extensively in the literature, using different models and notations [26, 28, 42, 43, 97, 101]. In 1876, Reuleaux [86] introduced the concept of kinematic chains. Almost one century later, in 1955, Denavit and Hartenberg [28] formalized a systematic notation for kinematic chains, which later became known as DH convention. In 1988, Veitschegger and Wu [108] extended the notation to describe complete geometric systems by adding base and tool transforms. The DH notation and it variants are still among the most used methods for identifying the kinematics of serial robots. A general introduction to the field of robotics, including a detailed description of the DH convention and its variants, can be found in the book by Siciliano and Khatib [98]; a comparison of different DH convention variants and an extensive study of geometric linkages can be found in book by McCarthy and Soh [74].

Barker [10] describes a vector-algebra approach for extracting geometric properties of assembled robotic arms. Rajeevlochana et al. [84] present a description for automatic model parameter estimation using a modified version of the algorithm based on line geometry. More details about their workflow, data acquisition, model extraction, and modeling error evaluation are given in the work of Hayat et al. [45] and in the work of Chittawadigi et al. [22].

The work mentioned above is not tailored to RTS, hence, device-specific algorithm steps and numerical optimization of extracted RTS models are not addressed by the authors. Furthermore, they do not mention downstream numerical optimization of the estimated models.

## 2.3 Simulation of Robots

Nowadays simulation of robotic devices and the interaction with the surroundings are widely used in research and engineering. This leads to a huge variety of programs and libraries for robot design and simulation, such as USARSim [20], Gazebo [68], Robocad [99], ABB RoboStudio [24], WorkcellSimulator [103], OpenSim [27], V-REP [87], Roboanalyzer [83]. However, some of these programs are commercially available only. Others are dedicated to specific robots and tasks that do not fit RTS well, or are even out-dated. Furthermore, heterogeneous systems, mobile clients and seamless exchange of simulator and hardware on top of an existing application programming interface (API) are not supported.

Mattingly et al. [73] describe the benefits of Unity3D for robot design and simulation, using Unity3D as the primary authoring tool. The work covers basic concepts such as design and implementation of the skeletal structure, geometry based on rigid bodies and animations that control object properties over time. While the work provides a good overview of concepts for robotic simulations, it does not describe the extraction of device models from existing hardware or a sufficient mathematical formulation.

Andaluz et al. [5] presented a Unity3D-based simulator for robotic teleoperation applications, which allows algorithm development in MATLAB. Similar to Alexandrescu et al. [2], the authors used *shared memory* for inter-process communication (IPC) to enable low latency data communication between the components. In their work, the authors model and simulate the robotic arm with kinematic and dynamic equations. Andaluz et al. provide a more detailed mathematical formalism of related robotic devices [4, 6]. The authors focus on simulating a robotic arm before physical construction and on developing low-level and closed-loop control systems. Furthermore, no analysis of the simulation uncertainty which is crucial for RTS applications is provided.

Meng et al. [77] introduced the real-time 3D simulation system *ROSUnitySim*, which can simulate multiple unmanned aerial vehicle (UAV) using the Robot Operation System (ROS) and Unity3D. This system allows experiments with ROS and multiple unmanned aerial vehicles, image sensors and light detection and ranging (LIDAR) sensors. Image and LIDAR sensors are modeled by Unity3D scripts, data packages between the individual components are routed by a Linux based *communication server* using TCP/IP sockets. The proposed simulator distributes computational expensive tasks, like ray casting based LIDAR data generation to multiple processes or workstations by using the network based client/server concept of Unity3D. The authors implementing an autonomous search task for multiple UAV to demonstrate the benefits of their system. A more complete description of the system is provided in later publication [49], but no details about geometric sensor and device modeling are included; multi-language bindings for heterogeneous systems are not directly supported. However, the aim of their work is algorithm development and simulation on top of ROS only.

Mattingly et al. [73] use Unity3D as primary authoring tool for robot design and simulation. Specifically, they talk about different topics related to rigid-body kinematics for skeletal structures, including a collection of authoring scripts and assets for Unity3D.

A simulator for tele-operated robots is described by Andaluz et al. [5]. Their work focuses on simulating closed-loop control systems with Unity3D and MATLAB. They use shared memory for exchanged data between the two programs to ensure low latency, and present promising results for path following and bilateral tele-operation tasks. As demonstration, the authors implemented an autonomous search task, with simultaneous localization and mapping (SLAM) and path planning for multiple drones. A more complete description of the system is provided in another publication [49], but no details about sensor and device modeling are included.

In this work, we provide a complete workflow for the design, implementation and characterization of RTS simulators. We also consider the simulation uncertainty of real-time robotic systems, which we think is essential for RTS simulation. Our methods are aimed to increase the efficiency, reliability and comparability of designing and testing interactive RTS simulators and algorithms. To the best of our knowledge, these aspects are not considered as major success factors in the work mentioned above and therefore have not been sufficiently addressed.

## 2.4 Visual Assistance for Manual Targeting

Many modern total stations are already equipped with one or more camera and image-based measurement methods, like steering the total station to selected pixels, selecting and visualizing 3D targets in the image or visualizing metadata. Scherer et al. [89, 91] investigated possible benefits of image-based features for architectural surveying. Zeiske [111] describes basic surveying methods and offline corrections for 2D corner measurements using simple geometry. However, no online method or image-based geometric correction are mentioned.

Ehrhart [31] provides a recent overview, concepts and applications for image-assisted total stations. He reviews related state-of-the-art approaches and selected patents, and discusses assessment methods for device comparison, calibration, monitoring and target tracking. In a previous work [32], the same author teamed up with colleagues to investigate image-processing methods for deformation monitoring. The authors detect movements of complete regions by comparing image patches acquired with the camera of a total station, without explicitly performing any structural analysis of building corners or edges.

Another vision-based system described by Siu et al. in [100] uses close range photogrammetric solution for 3D reconstruction and target tracking by combining several total stations and cameras.

Selected image-based modeling workflows for as-built visualization on construction sites are compared by Jadidi et al. in [56]. The authors reconstruct 3D point clouds and register as-built data to as-planned data. In particular, a case study on the image capturing approach should test the hypothesis whether or not an unordered photo collection, taken by field staff for regular progress report, is sufficient for image-based modeling and for visualization of as-built data. They compare the results to an guided recording approach, and conclude that the unordered photo collection is insufficient for visualization, as the capturing effort is not practicable, the registration success core of images is generally low, and only sparse point clouds are generated.

Fathi et al. [36] generate 3D wire diagrams of a roof using video streams from a calibrated stereo camera set. Their algorithm combines feature point matching, line detection and a-priori knowledge of roof structures to a structure-from-motion pipeline. Even if the results of these approaches are quite impressive, none of them can be applied for measuring corner and edge structures from a single position. Fathi et al. further note accuracy problems of the reconstructed models.

Closely related to our approach is the work by Juretzko [60], who provides conceptional descriptions for not directly measurable targets using intersections of 3D rays, lines and planes. However, no comparative study of the methods, no detailed mathematical description and no suitable user interface are provided. Furthermore, the author mentions only minimal sets of measured points for each method without any model fitting approach.

## 2.5 Assistance for Reflectorless Registration

The ICP algorithm and it variants are widely used practical solutions to the object registration problem. The algorithm was first introduced by Chen and Medioni [21], and Besl and McKay [14]. Mehdi [76] provides a comprehensive study of rigid body registration and error evaluation using the ICP algorithm. However, to the best of our knowledge, no complete system for the registration of an RTS with respect to a CAD model was proposed so far. We describe robust initialization and ICP sample selection for the proposed problem and an optimization for the measurement order.

In our work, local Manhattan-like structure detection and pose estimation is used for initializing the registration flow. The low measurement rate of RTS requires a robust algorithm for sparse point clouds. Hulik et al. [50] present a robust plane extraction algorithm based on the Hough transform [48]; Schnabel et al. [93] use the Random Sample Consesus (RANSAC) framework for robust shape extraction. However, both methods do not perform well for the desired point cloud sparsity. Our algorithm is based on the plane estimation algorithm proposed by Nguyen et al. [79]. The authors combine a mobile EDM with a real-time SLAM system to extract 3D models from sparse 3D point measurements. In particular, the authors apply graph-based point segmentation and Expectation Maximization (EM) plane fitting. We modified the proposed method for RTS and realistic indoor scenarios, for which image-based segmentation is not applicable due to the frequently occurring low contrast regions.

## 2.6 Discussion

The literature listed above shows a clear trend towards more intelligent RTS. We think that the simulation and integration of the assistance systems proposed in the following chapters is a valuable contribution in this field.

Compared to the work mentioned above, our methods are suitable for a wider range of devices, as we explicitly avoid photogrammetry, high-resolution imaging, high frame rates, and the use of an overview camera. This allows us to use parallax-free RTS systems with a single camera only.

We assume that the reader is familiar with the literature listed in Figure 2.1. This is not absolutely necessary as this work contains all required information and references to understand the proposed algorithms. However, knowing the fundamentals about surveying, total stations and camera models would undoubtedly be beneficial for reading this work and to apply the proposed workflows to practical issues with comparable properties.

In particular, camera and device calibration are handled extensively in literature and are therefore not reviewed in this work. For general camera models and calibration, the reader is referred to the book by Hartley and Zisserman [44]. Calibration and models of camera-assisted RTS with state-of-the-art approaches are explicitly handled in the book by Ehrhart [31].

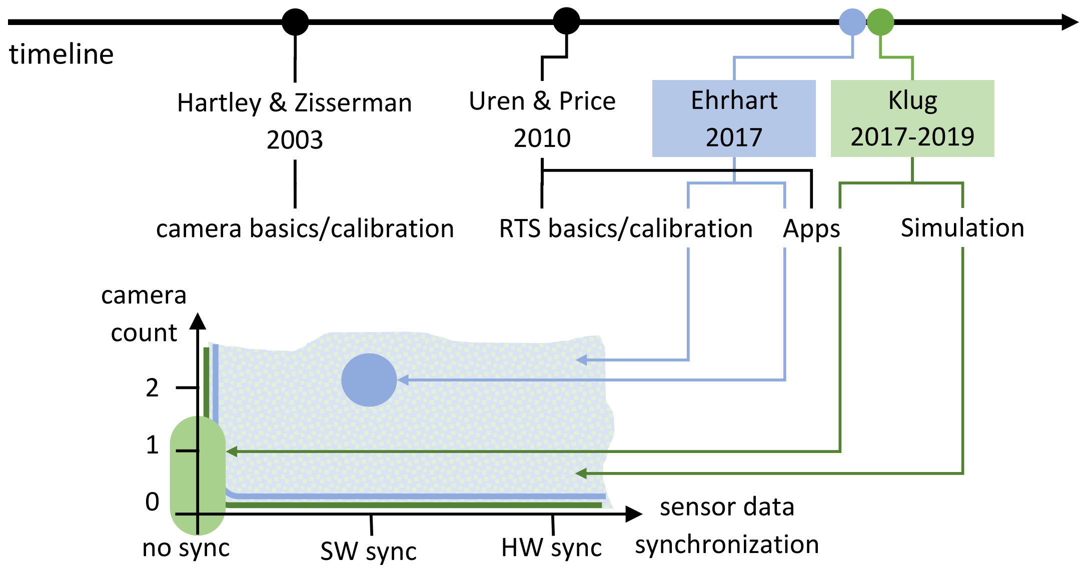


Figure 2.1: Recommended RTS literature, related topics and hardware coverage. Both the available camera count and the synchronization type between sensor data are crucial for hardware and algorithm selection: Algorithms with lower hardware requirements can be used for devices with higher camera count or better sensor data synchronization. Adjusting methods for devices with less cameras or worse sensor data synchronization is often expensive, complex, expensive or not possible at all. To address a wide range of devices, we focus on devices with one camera and no explicit sensor data synchronization in this work.

# 3 A Robotic Theory for RTS Modeling and Simulation

Data set generation for RTS algorithms is an expensive and time-consuming task, particularly, if special facilities or conditions are required. In this work, we use a novel virtual device simulator to avoid hardware-in-the-loop configurations in early design phases. This requires a kinematic model of the RTS to describe the geometry of motion for the mechanical system. In particular, a kinematic model describes the geometric transformations between rigid parts and various end-effectors. Here, the relationship between robot control parameters and geometric transformation matrices must be established. Figure 3.1 shows the general relationship between the control parameter space and the Cartesian space in the context of robot kinematics.

In surveying and measurement research, detailed kinematic device models are often not available due to company secrets or missing documentations.

In the following, we present a complete workflow for extracting kinematic models for real-time simulations of physical RTS setups and for verifying the simulation uncertainty. Our concept allows designing and testing interactive RTS algorithms with a virtual device and verifying the algorithm integrity with physical setups without further software modifications. In particular, the following sections describe

1. a method to extract forward kinematic models from RTS or devices with similar sensor configurations,
2. the analysis of the model discrepancy with respect to an RTS device driver,
3. the simulation of RTS sensors, actuators and uncertainties with respect to manufacturer specifications, and
4. the uncertainty characterization of different simulation setups according to Guide to the Expression of Uncertainty (GUM).

This chapter is based the work by Klug et al. [63, 66], but provides a more complete description.

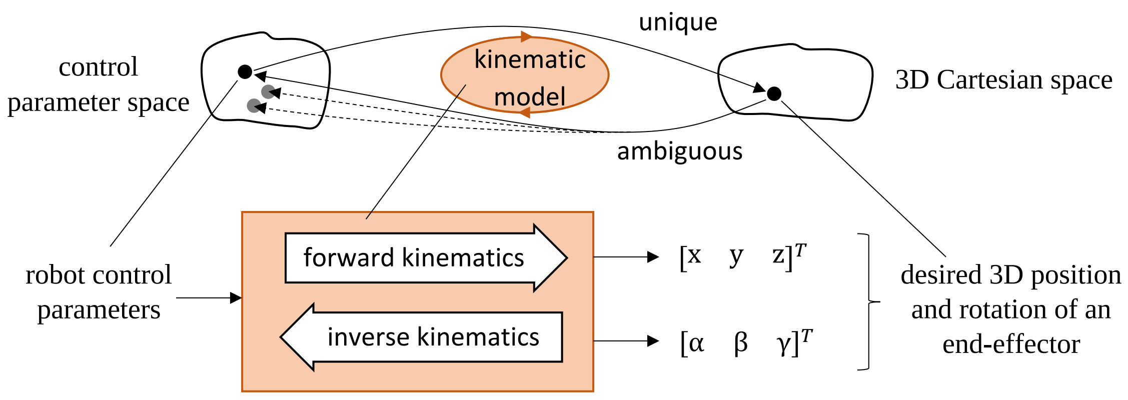


Figure 3.1: Relationship between control parameter space and Cartesian space of a robotic system.

## 3.1 Forward Kinematic Modeling

In the following, we model an RTS as electrical theodolite with one EDM and one camera, as shown in Figure 1.1. Hereby, the camera replaces the eyepiece of traditional devices [31, 106]. The idealized model defines a spherical coordinate system, but does not consider inevitable inaccuracies in manufacturing and physical device calibration.

Extended geometric models and software-side calibrations can reduce these errors, but the required manufacturing details are usually trade secrets and not available to the public. In general, the geometric model used internally by a device driver can be considered as inaccessible.

RTS coordinate systems are not standardized and can vary between manufacturers and device types. Therefore, we use commonly available low-level RTS functions to extract a combined model for hardware and software. This leads to a generic RTS model estimation approach, which allows modeling a variety of devices without the need for in-depth knowledge about the particular physical device.

The linear geometric properties of the rigid parts of an RTS can be described as spatial linkage. A general description is a series of Euclidean transformation matrices with six degree of freedom (DOF) each, as shown in Fig. 3.2. The geometric representation of an RTS can be converted to an open kinematic chain with fewer DOF for applying systematic model estimation based on robotic theory [66]. In this work, we use the DH convention [28] to describe an RTS with *links* and *joints* as shown in Fig. 3.3. The DH convention was originally introduced to describe the geometric relationship of open kinematic chains of $M-1$ joints and $M$ links by a series of homogeneous joint displacement matrices $\mathbf{Z}_i$ and link displacement matrices $\mathbf{X}_i$ [28].

In this work, we use the DH notation as proposed by Rajeevlochana and Saha [83, 88]. The convention defines that joints are rigidly connected by links and controllable at runtime with *joint control variables*. Note that this model ignores the dynamics of actuators,

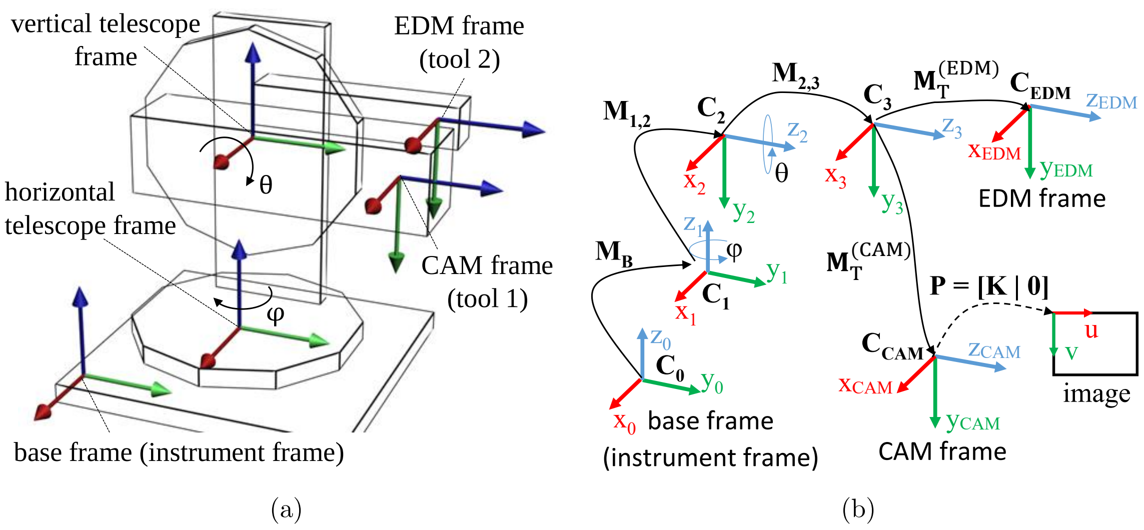


Figure 3.2: Extended geometric model for an exemplary RTS. For better visualization, the position of the frames are drawn non-overlapping. (a) Identifying frames. (b) Identifying transformations.

driving forces or moments. The geometric representation of the RTS shown in Fig. 3.3c consists of two controllable *revolute joints* and thereby is called RR-chain [69].

By convention, joint $i$ connects link $i-1$ and link $i$. The displacement matrices $\mathbf{Z}_i$ and $\mathbf{X}_i$ define the local coordinate frame $i$ by $(\mathbf{o}_i, \mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i)$, where $\mathbf{o}_i$ is the frame origin, and $\mathbf{x}_i$, $\mathbf{y}_i$, and $\mathbf{z}_i$ are the normalized $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ axes, respectively. Frame $i$ is rigidly attached to the end of link $i-1$ and must satisfy the following conditions:

1. The $\mathbf{x}_i$-axis is perpendicular to the $\mathbf{z}_{i-1}$-axis, and
2. the $\mathbf{x}_i$-axis intersects with the $\mathbf{z}_{i-1}$-axis.

The DH convention defines two types of joints:

1. *revolute joint*
2. *prismatic joint*

A revolute joint allows a rotation around the $\mathbf{z}$-axis of frame $i$ by the angle $\gamma_i$. A prismatic joint allows the translation along the $\mathbf{z}$-axis of frame $i$ by the distance $d_i$.

Figure 3.3d shows the relative pose $\mathbf{M}_{i,i+1}$ of frame $j$ with respect to frame $i$ using the DH convention, which is given by

$$\mathbf{M}_{i,j} = \prod_{n=i}^{j} \mathbf{M}_{n,n+1} \qquad \mathbf{M}_{i,i+1} = \mathbf{Z}_i \cdot \mathbf{X}_i \tag{3.1}$$

The joint matrix $\mathbf{Z}_i$ describes a screw displacement along the $\mathbf{z}$-axis of frame $i$. The link matrix $\mathbf{X}_i$ describes a rigid screw displacement along the $\mathbf{x}$-axis of frame $i$. The

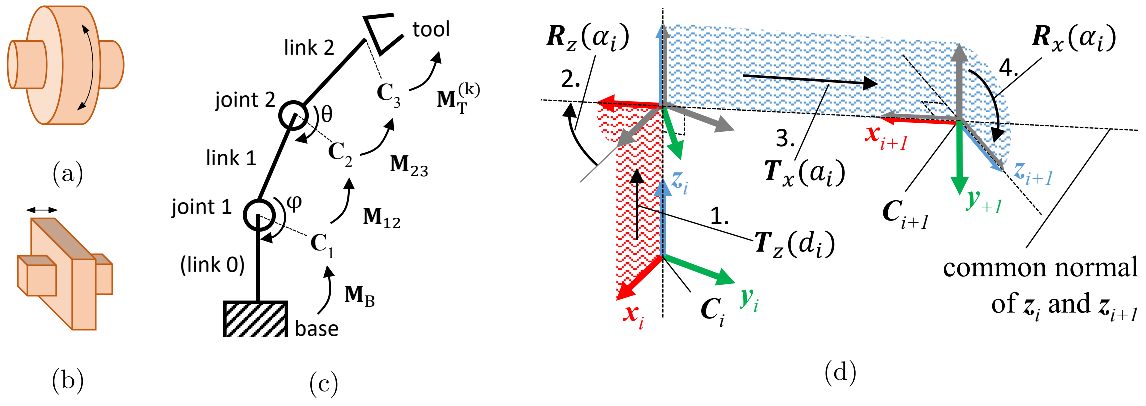


Figure 3.3: (a) Revolute joint. (b) Prismatic joint. (c) General kinematic chain of RTS. (d) Relative frame transformations, defined by the DH. The convention used in this work defines matrix $\mathbf{M}_{i,i+1}$ as the relative pose of frame $i+1$ with respect to frame $i$.

displacement matrices are given by

$$\mathbf{Z}_i = \mathbf{T}_z(d_i) \cdot \mathbf{R}_z(\gamma_i) \qquad \mathbf{X}_i = \mathbf{T}_x(a_i) \cdot \mathbf{R}_x(\alpha_i) \qquad Q_i = (d_i, \gamma_i, \alpha_i, a_i) \tag{3.2}$$

where the *DH parameters* are the elements of the parameter set $Q_i$. Hereby, $d_i$ and $\gamma_i$ are the dynamic control variables for joint $i$. The static properties of link $i$ are defined by $a_i$ and $\alpha_i$. The rigid twist of link $i-1$ is given by the rotation matrix $\mathbf{R}_x$, the rigid length of link $i-1$ is given by the translation matrix $\mathbf{T}_x$. The rotation of link $i$ around joint $i$ is given by the rotation matrix $\mathbf{R}_z$, the translation of joint $i$ along the $\mathbf{z}_i$-axis is given by the translation matrix $\mathbf{T}_z$. Rotation and translation matrices are defined in Eqn. A.4 and Eqn. A.6, respectively.

Finally, the pose of frame $i+1$ with respect to frame $i$ can be written as

$$\mathbf{M}_{i,i+1} = \begin{bmatrix} \cos\gamma_i & -\sin\gamma_i\cos\alpha_i & \sin\gamma_i\sin\alpha_i & a_i\cos\gamma_i \\ \sin\gamma_i & \cos\gamma_i\cos\alpha_i & -\cos\gamma_i\sin\alpha_i & a_i\sin\gamma_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} = \left[\begin{array}{ccc|c} & \mathbf{R}_{i,i+1} & & \mathbf{t}_{i,i+1} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{3.3}$$

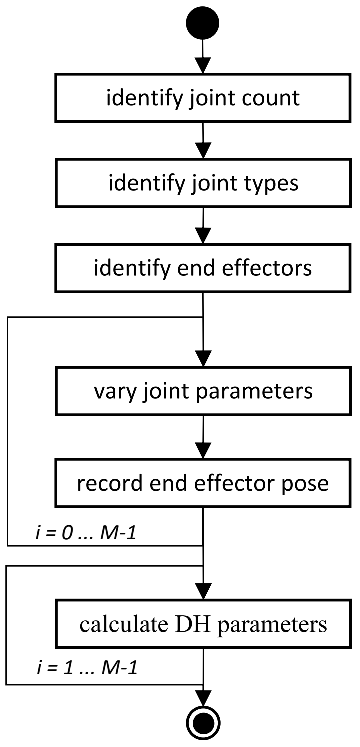
where $\mathbf{R}_{i,i+1}$ is a 3x3 rotation matrix, and $\mathbf{t}_{i,i+1}$ is a 3x1 translation vector.

## 3.1.1 Workflow

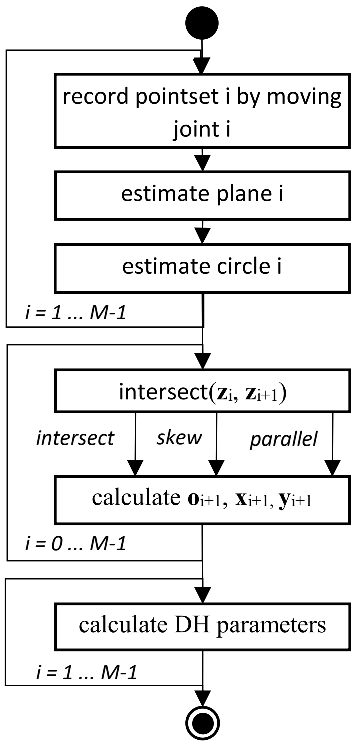
Estimating the forward kinematic model of an RTS can be reduced to the problem of deriving the DH parameters for a robotic device.

Figure 3.2a shows the coordinate frames of an RTS with a single camera[1]. The pose of the RTS with respect to the reference frame $\mathbf{C}_0$ is defined by the base transform $\mathbf{M}_B$ to allow arbitrary placement of the robot in the scene. The rigid attachment of different

[1]Coordinate frames for different RTS are not standardized and vary between individual devices.



(a)



(b)

Figure 3.4: (a) General workflow for extracting DH parameters of a serial kinematics chain [22, 45]. (b) Detailed workflow for extracting DH parameters of an RTS.

tools to the last link is described by individual tool transforms $\mathbf{M}_T^{(k)}$. The transformation of homogeneous points from tool space $k$ to the reference space is given by

$$\hat{\mathbf{x}}_{i,0} = \mathbf{M}_{B,T}^{(k)} \hat{\mathbf{x}}_{i,T}^{(k)} \qquad \mathbf{M}_{B,T}^{(k)} = \mathbf{M}_B \cdot \prod_{n=1}^{M-1} \mathbf{M}_{n,n+1} \cdot \mathbf{M}_T^{(k)} \tag{3.4}$$

where $\mathbf{M}_{B,T}^{(k)}$ is the pose of tool $k$ with respect to the reference frame, $\mathbf{M}_{n,n+1}$ is the pose of frame $n+1$ with respect to frame $n$ as given in Eqn. 3.1, $\hat{\mathbf{x}}_{i,T}^{(k)}$ defines a homogeneous point in the tools space $k$, and $\hat{\mathbf{x}}_{i,0}$ defines the same point in the reference space.

The RTS model shown in Fig. 3.3c can be fully described by a reference frame $\mathbf{C}_0 = \mathbf{C}_B$, two revolute joints, and two end-effectors, $\mathbf{C}_T^{(\mathrm{EDM})}$ and $\mathbf{C}_T^{(\mathrm{CAM})}$.

The first and second joint describe the horizontal rotation $\varphi$ and the vertical rotation $\theta$, respectively. The EDM frame $\mathbf{C}_T^{(\mathrm{EDM})}$ and the camera frame $\mathbf{C}_T^{(\mathrm{CAM})}$ describe the rigidly attached end-effectors. Note that only the end-to-end transformations between base frame and tool frames matches the device manual, the inner frames of the model are defined differently by the DH convention.

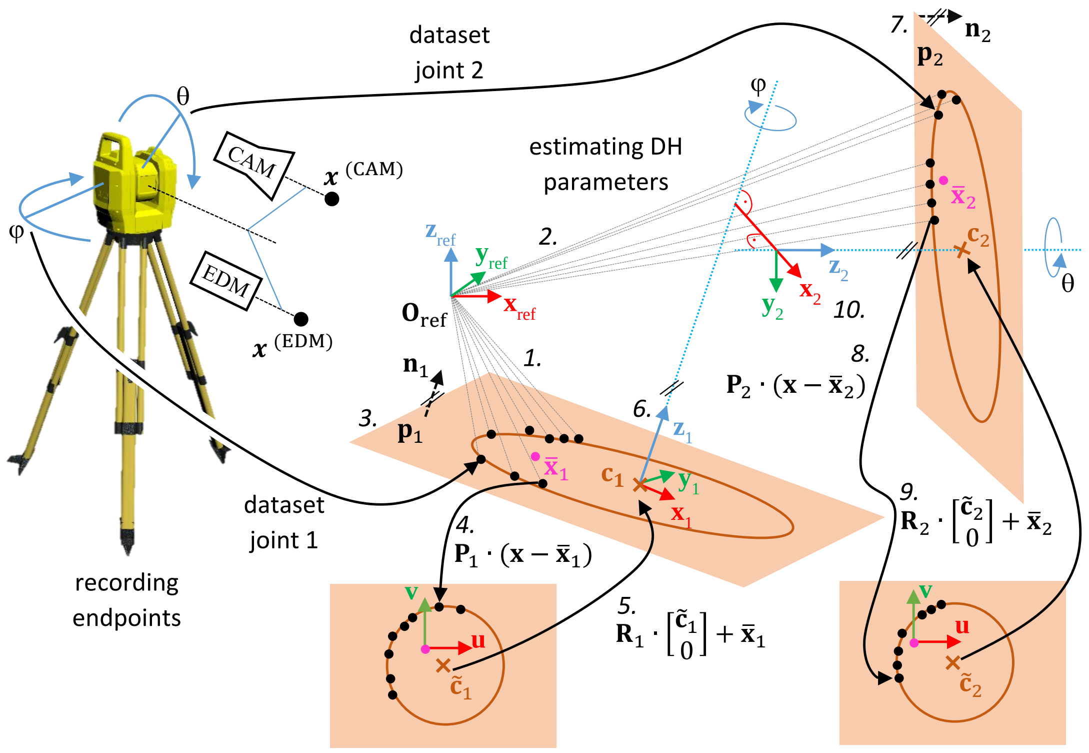


Figure 3.5: Workflow for extracting circular features of the RTS joints. The following steps are required: 1,2: measure or read out sensor points; 3,7: fit plane to each point cloud individually; 4,8: project points of each point cloud to 2D, using the extracted plane as transformation; 5,9: fit 2D circle and transform result to circle in 3D; 6,10: define frames and DH parameters.

To estimate the rotation and orientation of the joints, we record end-effector positions with respect to the reference frame, while rotating only one joint at a time. Figure 3.4a shows the general workflow for estimating the DH parameters of a serial kinematic chain according to Chittawadigi et al. [22, 45]. Figure 3.4b shows the modified workflow for measuring end-effector positions and extracting DH parameters of an RTS.

The recorded 3D points describe a planar circular trajectory. The center of the circles and the plane of rotations define a circular feature for each joint, which are used to estimate the DH parameters. Figure 3.5 shows the circular feature extraction workflow.

## 3.1.2 Data Acquisition for DH Parameter Estimation

The estimation of DH parameters usually requires an external measurement setup for measuring the pose of the end-effectors. However, for forward kinematic model estimation of an RTS, the required data can be fetched from the API of the device without any

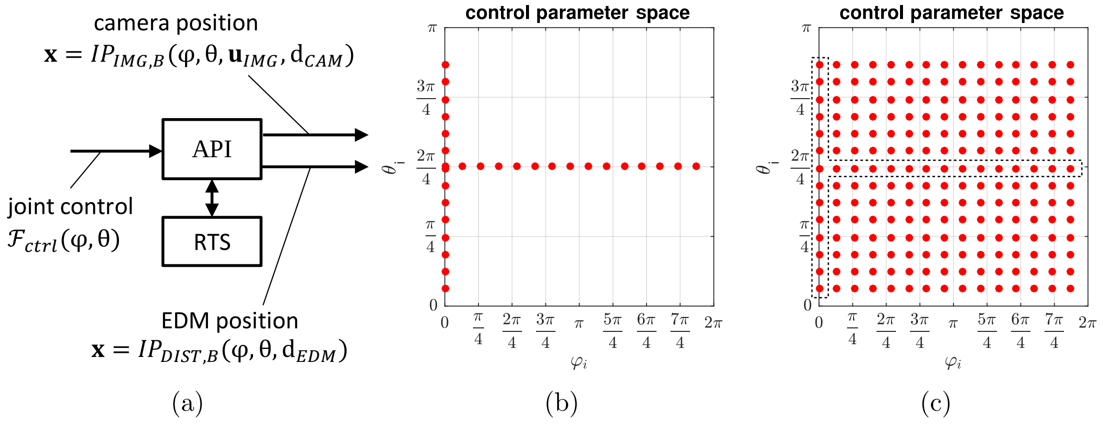

camera position
$\mathbf{x} = IP_{IMG,B}(\varphi, \theta, \mathbf{u}_{IMG}, d_{CAM})$
API
RTS
joint control
$\mathcal{F}_{ctrl}(\varphi, \theta)$
EDM position
$\mathbf{x} = IP_{DIST,B}(\varphi, \theta, d_{EDM})$
(a)
control parameter space
$\theta_i$
$\varphi_i$
(b)
control parameter space
(c)

Figure 3.6: (a) Required API and control parameters for converting or recording 3D point sets for each RTS joint and each end-effector. The exemplary device has two end-effectors, namely an EDM and a camera. (b) A subset of the angle control parameter space is sufficient to estimate the DH model. (c) Discrete samples of the full angle control parameter space, which is used for model cross-validation. Critical regions are excluded from both estimation and validation. Such regions include parameter ranges with expected singularities, model ambiguities or non-measurable areas, as shown Fig. 1.1.

external devices[2].

For each joint, a trajectory is defined by recording end-effector positions, while varying the related joint control parameter and keeping other control parameters constant. A linear trajectory of the recorded end-effector positions indicates a prismatic joint. A circular trajectory of the recorded points indicates a revolute joint.

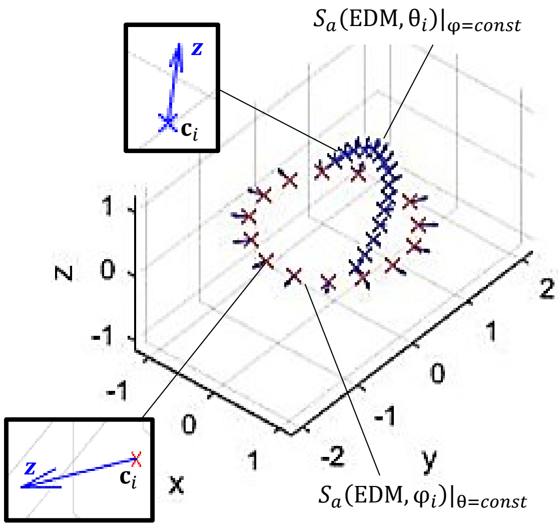
If the recorded data does neither describe a linear or circular movement, the affected joint type is either not prismatic or revolute, or the end-effector coincides with the rotation axis of a revolute joint. To avoid singularities during DH parameter extraction, the recording for the affected joint has to be repeated using different fixed joint settings or different API parameters[3].

Figure 3.6a shows the required measurement API. Joint $i = 1$ is the horizontal rotation, joint $i = 2$ is the vertical rotation, and $F(\varphi, \theta)$ is the control function for the corresponding joints. Data recording for joint 1 can be done by varying $\varphi$ in the range $[0, 2\pi]$, while keeping $\theta = \frac{\pi}{2}$ constant. Data recording for joint 2 can be done by varying $\theta$ between $[0, \pi]$, while keeping $\varphi = 0$ constant. The device API function

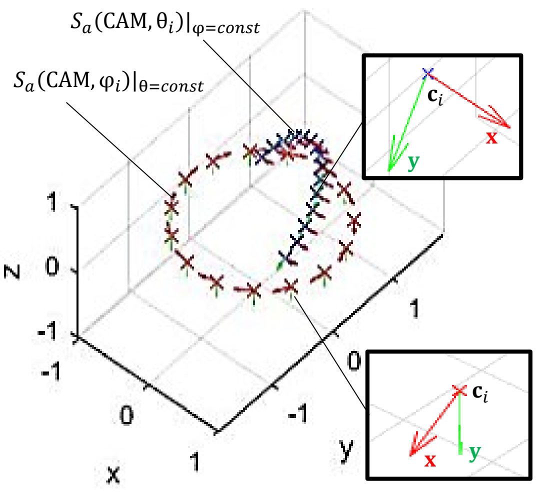
$$\mathbf{x}_i = IP_{DIST,B}(\varphi_i, \theta_i, d_{\mathrm{EDM}}) \tag{3.5}$$

applies the control variables $(\varphi_i, \theta_i)$ and converts a 1D distance $d_{\mathrm{EDM}}$ from EDM space to

[2]For device calibration, external measurements would still be required.
[3]If all link lengths are close to zero, an artificial end-effector offset must be applied. This can be done by recording 3D points that do not coincide with the end-effector origin.



(a) EDM positions



(b) camera positions

Figure 3.7: (a) Recorded end-effector positions $\mathbf{c}_i$ of the EDM. The EDM direction is determined by applying an artificial EDM distance measurement of one meter to each recorded position. In general, the remaining rotation of the EDM frame around the ray can be chosen freely. (b) Recorded end-effector positions $\mathbf{c}_i$ of the camera. The camera orientation is determined from pixel offsets.

to a 3D point $\mathbf{x}_i$ in the reference frame. The device API function

$$\mathbf{x}_i = IP_{IMG,B}(\varphi_i, \theta_i, \mathbf{u}, d_{\text{CAM}}) \tag{3.6}$$

applies the control variables $(\varphi_i, \theta_i)$ and converts a 2D image pixel $\mathbf{u}$ to a 3D point $\mathbf{x}_i$ in the reference frame. The two functions $IP_{DIST,B}$ and $IP_{IMG,B}$ are sufficient for forward kinematic modeling. They must be provided by the API of the RTS. Figure 3.6b shows the parameter space of the angle control variables $(\varphi_i, \theta_i)$; Figure 3.7a and Figure 3.7b show the positions of the recorded EDM and camera end-effectors, respectively.

### 3.1.3 Estimating Circular Features

A plane $\mathbf{p}$ can be defined according to Eqn. A.8, where a point on the plane $\bar{\mathbf{x}}$ is calculated from the center of mass according to Eqn. A.7.

The plane normal $\mathbf{n}$ can be fitted to the point cloud $S_p$ using Singular Value Decomposition (SVD) according to Appendix A.3.1.

It is convenient to define a right-handed orthogonal basis $\mathbf{B}$ for each plane such that the plane normal is aligned with the $\mathbf{z}$-axis, and the plane contains two additional orthogonal vectors according to

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det(\mathbf{U}) \end{bmatrix} \mathbf{U} \qquad \mathbf{B} = \begin{bmatrix} \mathbf{b}_1 & \mathbf{b}_2 & \mathbf{b}_3 \end{bmatrix} \qquad \mathbf{n} = \mathbf{b}_3 \tag{3.7}$$

The determinant of $\mathbf{U}$ is either equal to 1 if no reflection happens, or, equal to $-1$ in case of a reflection. Therefore, $\mathbf{B}$ can be interpreted as a reflection-free rotation of the plane with respect to the reference frame.

All points must be transformed to the $xy$ plane and projected to a 2D Euclidean space before fitting a circle to the planar measurements. This can be formalized as projection of the centered measurement matrix $\bar{\mathbf{A}}$ given in Eqn. A.13:

$$\mathbf{A}_c = \mathbf{P}_c \cdot \bar{\mathbf{A}} \qquad \mathbf{P}_c = \begin{bmatrix} \mathbf{b}_1^T \\ \mathbf{b}_2^T \end{bmatrix} \tag{3.8}$$

where $\mathbf{A}_c$ is a $[2 \times N]$ matrix, and $\mathbf{P}_c$ is a $[2 \times 3]$ projection matrix, which applies the inverse plane rotation $\mathbf{B}^T$ and the projection of the stacked 3D points $\bar{\mathbf{A}}$ to the 2D space[4].

A circle in 2D is given by the implicit equation

$$(u_i - c_1)^2 + (v_i - c_2)^2 = r^2 \tag{3.9}$$

where $r$ is the radius of the circle, $\tilde{\mathbf{c}} = \begin{bmatrix} c_1 & c_2 \end{bmatrix}^T$ is the center of the circle, and $\begin{bmatrix} u_i & v_i \end{bmatrix}^T$ is a point on the circle in 2D. Rearranging Eqn. 3.9 leads to

$$2 \cdot u_i \cdot c_1 + 2 \cdot v_i \cdot c_2 + k_3 = u_i^2 + v_i^2 \qquad k_3 = r^2 - c_1^2 - c_2^2 \tag{3.10}$$

which is linear in the unknown parameters $c_1$, $c_2$ and $k_3$. This can be written as an inhomogeneous linear system

$$\tilde{\mathbf{A}}_c^T \cdot \mathbf{x}_c = \mathbf{b}_c \qquad \tilde{\mathbf{A}}_c = \begin{bmatrix} 2 \cdot \mathbf{A}_c \\ \mathbf{1}_{1 \times N} \end{bmatrix} \qquad \mathbf{x}_c = \begin{bmatrix} c_1 \\ c_2 \\ k_3 \end{bmatrix} \qquad \mathbf{b}_c = \begin{bmatrix} \xi_1 \\ \xi_2 \\ \cdots \\ \xi_N \end{bmatrix} \qquad \xi_i = u_i^2 + v_i^2 \tag{3.11}$$

where $\tilde{\mathbf{A}}_c$ is a $3 \times N$ matrix, and $\xi_i$ is the sum of squared 2D coordinate values of point $i$. SVD can be used to solve for the unknown circle parameters:

$$\mathrm{SVD}(\tilde{\mathbf{A}}_c^T) = \mathbf{U}_c \mathbf{S}_c \mathbf{V}_c^T \qquad \mathbf{x}_c = \mathbf{V}_c \mathbf{S}_c^+ \cdot \mathbf{U}_c^T \mathbf{b}_c \qquad \mathbf{S}_c^+ = \left[ \begin{array}{ccc|c} 1/S_{c,11} & 0 & 0 & \\ 0 & 1/S_{c,22} & 0 & \mathbf{0}_{3 \times (N-3)} \\ 0 & 0 & 1/S_{c,33} & \end{array} \right] \tag{3.12}$$

[4]No homogeneous coordinates are required, since $\mathbf{P}_c$ does not include any 3D translation.

where $\mathbf{U}_c$ is a $N \times N$ matrix, $S_c$ is a $N \times 3$ matrix and $\mathbf{V}_c^T$ is a $3 \times 3$ matrix.

The 3D circular feature $\Phi_c$ for a single joint is fully defined by $\Phi_c = (\mathbf{c}, r)$, using

$$\mathbf{c} = \mathbf{B} \cdot \begin{bmatrix} c_1 \\ c_2 \\ 0 \end{bmatrix} + \bar{\mathbf{x}} \qquad r = \sqrt{k_3 + c_1^2 + c_2^2} \tag{3.13}$$

where $\mathbf{B}$ is the plane rotation according to Eqn. 3.7; $\mathbf{c}$ is the center of the circle with respect to the reference frame, and $r$ is the radius of the circle.

### 3.1.4 Correcting the Sign of Circular Features

A plane normal $\mathbf{n}$ as given in Eqn. A.8 is defined up to sign. This leads to sign ambiguities in the DH control parameters. For revolute joints, this can lead to an inverse rotation; for prismatic joints, this can lead to an inverse translation. The ambiguity can be resolved using two consecutive measurement points $\mathbf{x}_1$ and $\mathbf{x}_2$ of the circular feature $i$

$$\tilde{\mathbf{n}}_i = \operatorname{sign}\left( ((\mathbf{x}_1 - \mathbf{c}_i) \times (\mathbf{x}_2 - \mathbf{c}_i))^T \, \mathbf{n}_i \right) \mathbf{n}_i \tag{3.14}$$

where $\mathbf{c}_i$ is the center of the circle, and $\operatorname{sign}(x)$ is the sign of the scalar value $x$ according to Eqn. A.1. The corrected plane normal $\mathbf{n}_i$ ensures a positive rotation direction for increasing control parameter values, if point $\mathbf{x}_1$ was recorded with a smaller control parameter than point $\mathbf{x}_2$[5]. Figure 3.8a shows the sign correction of circular features.

### 3.1.5 Link Constellation and Frame Alignment

The link constellation and frame alignment are based on the spatial relationship of the $\mathbf{z}_{i-1}$ and the $\mathbf{z}_i$ axes. An intersection is treated as special case of skewed lines.

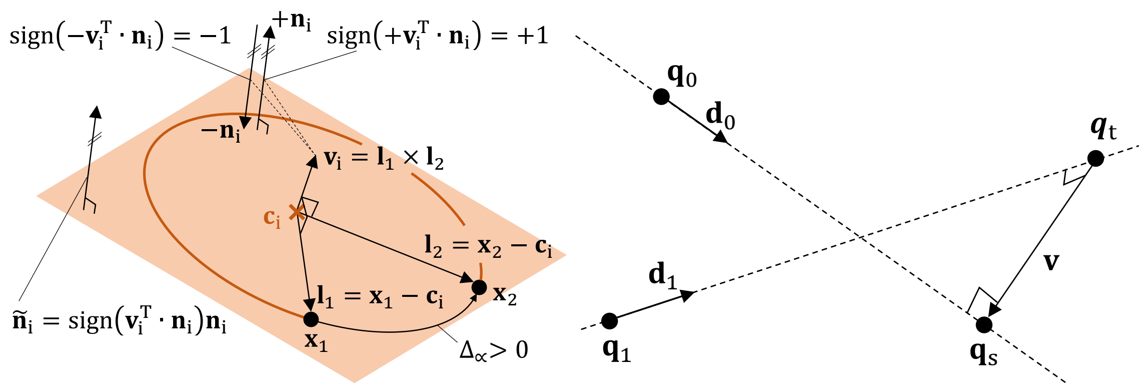
Rajeevlochana et al. used Plücker coordinates and Dual Vector Algebra for estimating the line constellation [84]. Plücker coordinates allow closed-form line intersection testing. However, for kinematic chains with low link counts, we did not observe any computational benefits when using Plücker coordinates instead of simple vector algebra as described in the work by Barker [10]. Thus, for the sake of simplicity, we describe line constellations using simple vector algebra.

Figure 3.8b shows the distance of two lines in the 3D space. Given two lines $\mathcal{L}_0$ and $\mathcal{L}_1$ in their parametric form

$$\mathcal{L}_0(s) = \mathbf{q}_0 + s\mathbf{d}_0 \qquad \mathcal{L}_1(t) = \mathbf{q}_1 + t\mathbf{d}_1 \tag{3.15}$$

we wish to find the minimum distance between the two lines. Let $\mathbf{q}_s$ and $\mathbf{q}_t$ define points

[5]For revolute joints, it is advisable to use control angles between $[0, \pi]$ for sign correction to avoid errors caused by periodicity.


$\mathrm{sign}(-\mathbf{v}_i^T \cdot \mathbf{n}_i) = -1$
$+\mathbf{n}_i$
$\mathrm{sign}(+\mathbf{v}_i^T \cdot \mathbf{n}_i) = +1$
$-\mathbf{n}_i$
$\mathbf{v}_i = \mathbf{l}_1 \times \mathbf{l}_2$
$\mathbf{c}_i$
$\mathbf{l}_2 = \mathbf{x}_2 - \mathbf{c}_i$
$\mathbf{l}_1 = \mathbf{x}_1 - \mathbf{c}_i$
$\mathbf{x}_2$
$\mathbf{x}_1$
$\tilde{\mathbf{n}}_i = \mathrm{sign}(\mathbf{v}_i^T \cdot \mathbf{n}_i)\mathbf{n}_i$
$\Delta_\alpha > 0$
$\mathbf{q}_0$
$\mathbf{d}_0$
$\mathbf{q}_t$
$\mathbf{v}$
$\mathbf{d}_1$
$\mathbf{q}_1$
$\mathbf{q}_s$

(a) sign correction (b) 3D line geometry

Figure 3.8: (a) Algorithm for automatic sign correction for the circular features of the individual joints. The normal of the extracted planes $\pm\mathbf{n}_i$ is defined up to sign. The difference of the angular control parameters $\Delta_\alpha$ of two consecutive measurement samples $(\mathbf{x}_1, \mathbf{x}_2)$ defines the positive direction of rotation and the sign of corresponding the plane normal. (b) Minimal distance $||\mathbf{v}||$ between two 3D lines.

on line $\mathcal{L}_0$ and $\mathcal{L}_1$ that minimize the length of the vector

$$\mathbf{v} = \mathbf{q}_s - \mathbf{q}_t \qquad \mathbf{q}_s = \mathbf{q}_0 + s_s\mathbf{d}_0 \qquad \mathbf{q}_t = \mathbf{q}_1 + t_s\mathbf{d}_1 \tag{3.16}$$

The minimum Euclidean distance between the lines is given by $||\mathbf{v}||$, if $\mathbf{v}$ has the same direction as the common normal, which can be written as

$$\mathbf{d}_0^T\mathbf{v} = 0 \qquad \mathbf{d}_1^T\mathbf{v} = 0 \tag{3.17}$$

If we substitute Eqn. 3.16 in Eqn. 3.17 we can solve for the two unknown parameters $s_s$ and $t_s$

$$s_s = \frac{be - cd}{ac - b^2} \qquad t_s = \frac{ae - bd}{ac - b^2} \tag{3.18}$$

where

$$a = ||\mathbf{d}_0||^2 \qquad b = \mathbf{d}_0^T\mathbf{d}_1 \qquad c = ||\mathbf{d}_1||^2 \qquad d = \mathbf{d}_0^T(\mathbf{q}_0 - \mathbf{q}_1) \qquad e = \mathbf{d}_1^T(\mathbf{q}_0 - \mathbf{q}_1) \tag{3.19}$$

The parameters $s_c$ and $t_c$ for calculating the closest points $\mathbf{q}_s$ and $\mathbf{q}_t$ on line $\mathcal{L}_0$ and $\mathcal{L}_1$, respectively, are given by

$$s_c = \begin{cases} 0, & \text{parallel} \\ s_s, & \text{skewed} \end{cases} \qquad t_c = \begin{cases} d/b, & \text{parallel},\ b \geq c \\ e/c, & \text{parallel},\ b < c \\ t_s, & \text{skewed} \end{cases} \qquad c_c = \begin{cases} \text{parallel}, & (ac - b^2) = 0 \\ \text{skewed}, & \text{otherwise} \end{cases} \tag{3.20}$$

where the spatial constellation type of two 3D lines $c_c \in \{\text{parallel},\ \text{skewed}\}$ is given by the divisor of Eqn. 3.18.

Finally, the points $\mathbf{q}_s$ and $\mathbf{q}_t$ can be calculated by substituting the parameters from Eqn. 3.20 into the line equations given in Eqn. 3.15:

$$\mathbf{q}_s = \mathcal{L}_0(s_c) \qquad \mathbf{q}_t = \mathcal{L}_1(t_c) \tag{3.21}$$

### 3.1.6 First Link Frame

The $\mathbf{z}$ direction of the first frame is aligned with the plane normal of the first circular feature $\tilde{\mathbf{n}}_1$. The origin of the first frame $\mathbf{o}_1$ must lie on the first rotational axis $\mathcal{L}_{z1}(s) = \mathbf{c}_1 + s\tilde{\mathbf{n}}_1$; the $\mathbf{x}$-axis of the first frame must lie on the plane defined by $\mathbf{o}_1$ and $\mathbf{z}_1$. The rotation of $\mathbf{x}_1$ around $\mathbf{z}_1$ can be arbitrary defined. Hence, the first frame is fully defined by

$$\mathbf{z}_1 = \tilde{\mathbf{n}}_1 \qquad \mathbf{o}_1 = \mathbf{c}_1 \qquad \mathbf{B}_1 = \begin{bmatrix} \mathbf{x}_1 & \mathbf{y}_1 & \mathbf{z}_1 \end{bmatrix} \tag{3.22}$$

where $\mathbf{B}_1$ is a right-handed orthogonal base according to Eqn. 3.7.

However, this approach will lead to different DH parameters for different point sets of the first circular feature.

A more convenient approach is to further align the $\mathbf{x}$ direction of the reference frame $\mathbf{x}_0$ with the $\mathbf{x}$ direction of the first frame $\mathbf{x_1}$ by projecting $\mathbf{x}_0$ onto plane $\mathbf{p}_1$:

$$\mathbf{x}_1 = \frac{\mathbf{x}_0 - (\mathbf{x}_0^T \mathbf{z}_1)\mathbf{z}_1}{||\mathbf{x}_0 - (\mathbf{x}_0^T \mathbf{z}_1)\mathbf{z}_1||} \qquad \mathbf{y}_1 = \mathbf{z}_1 \times \mathbf{x}_1 \tag{3.23}$$

If plane $\mathbf{p}_1$ is parallel to the $yz$ plane of the reference frame, the alignment of the $\mathbf{y}$ directions can be used instead. Figure 3.9 shows the alignment of the first frame.

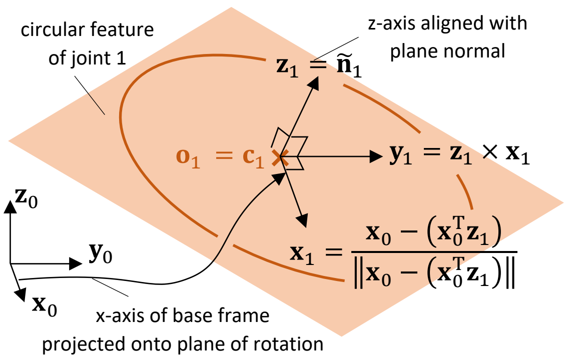

circular feature
of joint 1
z-axis aligned with
plane normal
$\mathbf{z}_1 = \tilde{\mathbf{n}}_1$
$\mathbf{o}_1 = \mathbf{c}_1$
$\mathbf{y}_1 = \mathbf{z}_1 \times \mathbf{x}_1$
$\mathbf{z}_0$
$\mathbf{y}_0$
$\mathbf{x}_0$
$\mathbf{x}_1 = \frac{\mathbf{x}_0 - (\mathbf{x}_0^T \mathbf{z}_1)}{\|\mathbf{x}_0 - (\mathbf{x}_0^T \mathbf{z}_1)\|}$
x-axis of base frame
projected onto plane of rotation

Figure 3.9: Alignment of first link frame.

### 3.1.7 Middle Link Frames

The frames $i = 2, \ldots, M-1$ can be defined iteratively using

$$\begin{bmatrix} \mathbf{z}_i \\ \mathbf{z}_{i+1} \\ \mathbf{o}_{i+1} \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{n}}_i \\ \tilde{\mathbf{n}}_{i+1} \\ \mathbf{q}_t^{(i)} \end{bmatrix} \qquad \mathbf{y}_{i+1} = \frac{\mathbf{z}_{i+1} \times \mathbf{x}_{i+1}}{||\mathbf{z}_{i+1} \times \mathbf{x}_{i+1}||} \qquad \mathbf{x}_{i+1} = \begin{cases} \frac{\mathbf{z}_i \times \mathbf{z}_{i+1}}{||\mathbf{z}_i \times \mathbf{z}_{i+1}||}, & \mathcal{L}_i \text{ and } \mathcal{L}_{i+1} \text{ intersect} \\ \frac{\mathbf{q}_t^{(i)} - \mathbf{q}_s^{(i)}}{||\mathbf{q}_t^{(i)} - \mathbf{q}_s^{(i)}||}, & \text{are skewed or parallel} \\ \mathbf{x}_i, & \text{are identical} \end{cases} \tag{3.24}$$

where $\mathbf{q}_s^{(i)}$ and $\mathbf{q}_t^{(i)}$ are the common normal intersections of $\mathcal{L}_i$ and $\mathcal{L}_{i+1}$ according to Eqn. 3.16, respectively. Based on the calculated coordinate frames, the DH parameters for a revolute joint $i$ can be derived by

$$\begin{bmatrix} b_i \\ \theta_i \\ a_i \\ \alpha_i \end{bmatrix} = \begin{bmatrix} (\mathbf{o}_{i+1} - \mathbf{o}_i)^T \mathbf{z}_i \\ \operatorname{arctan2}\left((\mathbf{x}_i \times \mathbf{x}_{i+1})^T \mathbf{z}_i,\ \mathbf{x}_i^T \mathbf{x}_{i+1}\right) \\ (\mathbf{o}_{i+1} - \mathbf{o}_i)^T \mathbf{x}_i \\ \operatorname{arctan2}\left((\mathbf{z}_i \times \mathbf{z}_{i+1})^T \mathbf{x}_{i+1},\ \mathbf{z}_i^T \mathbf{z}_{i+1}\right) \end{bmatrix} \tag{3.25}$$

### 3.1.8 Last Link Frame

The last frame can be used to define the tool pose. However, a general pose of the tool would require six DOF whereas a single frame of the DH framework is limited to four DOF. One solution to this problem is to use multiple DH frames to describe the tool pose. In this work, we used a more general approach by extracting a six DOF tool matrix $\mathbf{M}_T^{(k)}$ for each tool $k$ separately.

By including individual tool matrices, the rotation of the last link frame around the $\mathbf{z}_N$-axis is arbitrary. However, it is convenient to align the last frame with the previous one:

$$\begin{bmatrix}\mathbf{x}_M & \mathbf{y}_M & \mathbf{z}_M\end{bmatrix} = \begin{bmatrix}\mathbf{x}_{M-1} & \mathbf{y}_{M-1} & \mathbf{z}_{M-1}\end{bmatrix} \tag{3.26}$$

The DH parameters are then calculated according to Sec. 3.1.7.

### 3.1.9 Base Transform

The base transform defines the pose of the first coordinate frame, $(\mathbf{o}_1, \mathbf{x}_1, \mathbf{y}_1, \mathbf{z}_1)$, with respect to the reference frame, $(\mathbf{o}_0, \mathbf{x}_0, \mathbf{y}_0, \mathbf{z}_0)$, using the six DOF Euclidean transformation $\mathbf{M}_B$. This problem can be described as *estimating 3D rigid transformations between two ordered point sets* as defined in Appendix A.3.3, using

$$S_a = (\mathbf{c}_0 + \mathbf{x}_0, \mathbf{c}_0 + \mathbf{y}_0, \mathbf{c}_0 + \mathbf{z}_0) \qquad S_b = (\mathbf{c}_1 + \mathbf{x}_1, \mathbf{c}_1 + \mathbf{y}_1, \mathbf{c}_1 + \mathbf{z}_1) \tag{3.27}$$

$$(\mathbf{R}_B, \mathbf{t}_B) = F(S_a, S_b) \qquad \mathbf{M}_B = \left[\begin{array}{ccc|c} & \mathbf{R}_B & & \mathbf{t}_B \\ \hline 0 & 0 & 0 & 1\end{array}\right] \tag{3.28}$$

### 3.1.10 Camera Tool Frame

The pose of the end-effector $k$ with respect to the last link frame $(\mathbf{o}_M, \mathbf{x}_M, \mathbf{y}_M, \mathbf{z}_M)$ is defined by a six DOF rigid transform. If two ordered point sets with $N \geq 3$ correspondences can be obtained, the alignment algorithm as described in Sec. 3.1.9 can be applied.

The pose of the camera end-effector can be defined by the end-effector position, two orthogonal vectors describing the $\mathbf{x}$ and $\mathbf{y}$ direction, and the right-handed coordinate system constraint. The end-effector position was already used for the DH parameter extraction. The $\mathbf{x}$ and $\mathbf{y}$ directions of the camera are aligned with the $\mathbf{u}$ and $\mathbf{v}$ direction of the image space, respectively. By extending the ordered set of back-projected pixel coordinates during the measurement flow for the DH parameters, the $\mathbf{x}$ and $\mathbf{y}$ direction of the end-effector with respect to the reference frame can be extracted. Figure 3.7a and Figure 3.7b show the end-effector recordings, including the observable frame axis of the end-effectors; Figure 3.10a shows the back-projection method and the alignment of camera and image frame.

Given the camera intrinsics, the camera projection matrix $\mathbf{P}$ can be written as

$$\hat{\mathbf{u}}_i = \mathbf{P}_{\mathrm{CAM}} \mathbf{M}_{B,T}^{-1} \hat{\mathbf{x}}_i \qquad \mathbf{P}_{\mathrm{CAM}} = [\mathbf{K}|\mathbf{0}] \qquad \mathbf{K} = \begin{bmatrix} f_u & s & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1\end{bmatrix} \tag{3.29}$$

where $\hat{\mathbf{u}}_i$ is a homogeneous pixel point, $\hat{\mathbf{x}}_i$ is a homogeneous world-space point, $s$ is a skew factor, $(f_u, f_v)$ describe the focal lengths and $\mathbf{c} = \begin{bmatrix} c_u & c_v \end{bmatrix}^T$ describe the principal point offset in 2D[6].

[6]For the sake of simplicity, we set $s = 0$ and $f_u = f_v = f$, and we do not include lens distortion parameters.

Matrix $\mathbf{M}_{B,T}$ is the camera tool pose with respect to the reference frame, using $\mathbf{M}_T^{(k)} = \mathbf{M}_T^{(\mathrm{CAM})}$ in Eqn. 3.4. By defining the tool space as camera space, the relationship between a homogeneous point $\hat{\mathbf{x}}_T$ in the camera space and a homogeneous point $\hat{\mathbf{x}}_0$ in the reference space is given by

$$\hat{\mathbf{x}}_0 = \mathbf{M}_{B,T}\hat{\mathbf{x}}_T = \left(\mathbf{M}_B\mathbf{M}_{1,M}\mathbf{M}_T\right)\hat{\mathbf{x}}_T \tag{3.30}$$

where $\mathbf{M}_{1,M}$ is the relative pose of frame $M$ with respect to frame 1 as defined in Eqn. 3.4. Rearranging Eqn. 3.30 leads to

$$\mathbf{M}_{1,M}^{-1}\mathbf{M}_B^{-1}\hat{\mathbf{x}}_0 = \mathbf{M}_T\hat{\mathbf{x}}_T \tag{3.31}$$

where $\mathbf{M}_{1,M}$ and $\mathbf{M}_B$ are already known. The substitution of $\tilde{\mathbf{x}}_0 = \mathbf{M}_{1,M}^{-1}\mathbf{M}_B^{-1}\hat{\mathbf{x}}_0$ in Eqn. 3.31 leads to

$$\tilde{\mathbf{x}}_0 = \mathbf{M}_T\hat{\mathbf{x}}_T \tag{3.32}$$

which can be solved for $\mathbf{M}_T$ using $N \geq 3$ point correspondences. Finding point correspondences $\hat{\mathbf{x}}_T$ and $\tilde{\mathbf{x}}_0$ can be done using back-projection of pixels into the camera space.

Let $S_u = (\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3)$ be a ordered set of 2D image points using

$$\mathbf{u}_1 = \mathbf{c} \qquad \mathbf{u}_2 = \mathbf{c} + \begin{bmatrix}1\\0\end{bmatrix} \qquad \mathbf{u}_3 = \mathbf{c} + \begin{bmatrix}0\\1\end{bmatrix} \tag{3.33}$$

where $\mathbf{c}$ is the principal point of the camera. The image coordinates given in $S_u$ are back-projected by using the API function given in Eqn. 3.6

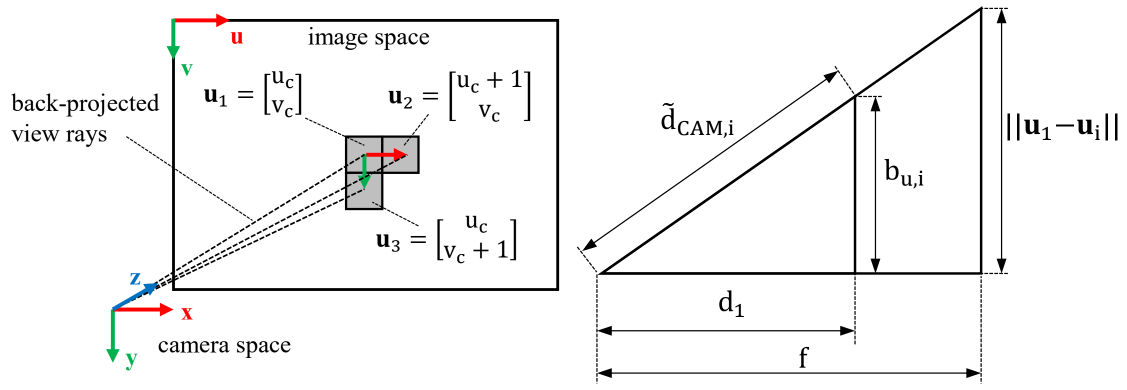
$$\mathbf{x}_i = IP_{IMG,B}(\varphi_j, \theta_j, \mathbf{u}_i, d_{\mathrm{CAM}}) \tag{3.34}$$

using $d_{\mathrm{CAM}} = 1$ and constant parameters $(\varphi_j, \theta_j)$ for the ordered set $S_u$.

The measured points are then converted to a local coordinate frame using $\mathbf{x}_1$ as origin

$$\begin{bmatrix}\hat{\mathbf{x}}_1 & \hat{\mathbf{x}}_2 & \hat{\mathbf{x}}_3\end{bmatrix}_{[4x3]} = \mathbf{M}_{1,M}^{-1}\mathbf{M}_B^{-1}\begin{bmatrix}\mathbf{x}_1 & \mathbf{x}_1 + \frac{\mathbf{x}_2-\mathbf{x}_1}{||\mathbf{x}_2-\mathbf{x}_1||} & \mathbf{x}_1 + \frac{\mathbf{x}_3-\mathbf{x}_1}{||\mathbf{x}_3-\mathbf{x}_1||}\\ 1 & 1 & 1\end{bmatrix}_{[4x3]} \tag{3.35}$$

The rigid transformation $\mathbf{M}_T^{(\mathrm{CAM})}$ can be extracted according to Sec. 3.1.6 using the two


u
image space
v
back-projected view rays
$\mathbf{u}_1 = \begin{bmatrix} u_c \\ v_c \end{bmatrix}$
$\mathbf{u}_2 = \begin{bmatrix} u_c + 1 \\ v_c \end{bmatrix}$
$\mathbf{u}_3 = \begin{bmatrix} u_c \\ v_c + 1 \end{bmatrix}$
z
x
y
camera space
$\tilde{d}_{CAM,i}$
$b_{u,i}$
$||\mathbf{u}_1 - \mathbf{u}_i||$
$d_1$
f

(a) back-projected pixels (b) view ray length

Figure 3.10: (a) Pixel back-projections used for estimating the orientation of the camera frame. (b) Relation between principal ray and back projected pixel used for validating the approximation effects on the camera frame orientation (simplified).

ordered point sets

$$S_a = \left( \mathbf{c}_t, \mathbf{c}_t + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \mathbf{c}_t + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \right) \qquad S_b = (\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \tilde{\mathbf{x}}_3) \tag{3.36}$$

where $\tilde{\mathbf{x}}_i$ is the Euclidean representation of the homogeneous coordinate $\hat{\mathbf{x}}_i$ given in Eqn. 3.35, and $\mathbf{c}_t$ is given by $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$.

A constant distance $d_{\text{CAM}} = 1$ for all back-projected points in Eqn. 3.34 will lead to a systematic error, which is negligible for the tool transform estimation. This can be shown by calculating the corrected length $d_{\text{CAM},i}$ of the back projected ray for the image coordinates $\mathbf{u}_i$ with $i \in \{2, 3\}$ in Eqn. 3.33. Assuming $\mathbf{u}_1$ is aligned with the principal ray and the focal length $f$ is known, the ray length $\tilde{d}_{\text{CAM},i}$ is given by triangle similarity and the Pythagorean theorem according to

$$\tilde{d}_{\text{CAM},i} = \sqrt{d_1^2 + b_{u,i}^2} \qquad \frac{b_{u,i}}{d_1} = \frac{||\mathbf{u}_1 - \mathbf{u}_i||}{f} \quad \Rightarrow \quad \tilde{d}_{\text{CAM},i} = d_1 \sqrt{1 + \left( \frac{||\mathbf{u}_1 - \mathbf{u}_i||}{f} \right)^2} \tag{3.37}$$

where $||\mathbf{u}_1 - \mathbf{u}_i||$ describes the distance of two pixels $(\mathbf{u}_1, \mathbf{u}_i)$ in the image space and $(b_{u,i}, d_1, \tilde{d}_{\text{CAM},i})$ define a right-angled triangle. For $f \gg ||\mathbf{u}_1 - \mathbf{u}_i||$ and $d_1 = d_{\text{CAM}} = 1$ we can set $\tilde{d}_{\text{CAM},i} \approx d_i$. Figure 3.10b shows the simplified relationship of back projected rays with respect to the principal ray.

### 3.1.11 EDM Tool Frame

Similar to the camera tool, the pose of the EDM end-effector can be described by a six DOF Euclidean transformation. Distance measurements can be modeled by a 3D ray in the Euclidean space. This leads to a tool pose defined up to an arbitrary rotation around the measurement axis. However, it is convenient to align the EDM frame with the camera frame.
The pose of the EDM end-effector can be defined by a ray that describes the distance measurement in 3D, a related orthogonal vector, and the right-handed coordinate system constraint. The end-effector position, which defines a point on the ray, was already used for the DH parameter extraction.

We define the **z**-axis of the EDM as the distance measurement direction. By extending the ordered set of back-projected distances during the measurement flow for the DH parameters, the **z** direction of the end-effector with respect to the reference frame can be extracted.

A 1D distance measurement can be back-projected using

$$\hat{\mathbf{x}}_i = \mathbf{M}_{B,T}^{(\mathrm{EDM})}\hat{\mathbf{d}}_i \qquad \mathbf{M}_{B,T}^{(\mathrm{EDM})} = \mathbf{M}_B\mathbf{M}_{1,M}\mathbf{M}_T^{(\mathrm{EDM})} \tag{3.38}$$

where $\hat{\mathbf{x}}_i$ is a homogeneous point in the reference frame, and $\hat{\mathbf{d}}_i$ is a homogeneous point in the EDM space. The homogeneous point $\hat{\mathbf{d}}_i$ for a distance measurement $d_i$ along the **z**-axis of the EDM frame is defined as

$$\hat{\mathbf{d}}_i = \begin{bmatrix} 0 & 0 & d_i & 1 \end{bmatrix}^T \tag{3.39}$$

Rearranging Eqn. 3.38 leads to

$$\mathbf{M}_{1,M}^{-1}\mathbf{M}_B^{-1}\hat{\mathbf{x}}_i = \mathbf{M}_T^{(\mathrm{EDM})}\hat{\mathbf{d}}_i \tag{3.40}$$

which is of the same form as Eqn. 3.31. We construct three correspondences for solving Eqn. 3.38 for the rigid transform $\mathbf{M}_T^{(EDM)}$. Let $S_d = (d_1, d_2)$ be a tuple of two distances where $d_1 = 1$ and $d_2 = 2$. The back-projection of the distances $d_i$ to 3D points in the reference frame can be done with the API function given in Eqn. 3.5

$$\mathbf{d}_i = IP_{DIST,B}(\varphi_j, \theta_j, d_i) \tag{3.41}$$

using constant parameters $(\varphi_j, \theta_j)$ for the tuple $S_d$. The **x** direction of the EDM frame can be found by projecting the **x** direction[7] of the camera frame onto the plane defined

[7]If the **x** direction of the camera frame and the **z** direction of the EDM frame are parallel, the **y** direction of the camera frame can be used instead.

by point $\mathbf{d}_1$ and plane normal $\mathbf{n} = \mathbf{z}_{EDM} = \frac{\mathbf{d}_2 - \mathbf{d}_1}{||\mathbf{d}_2 - \mathbf{d}_1||}$

$$\mathbf{x}_{\text{EDM}} = \frac{\mathbf{x}_{\text{CAM}} - (\mathbf{z}_{\text{CAM}}^T \mathbf{z}_{\text{EDM}})\mathbf{z}_{\text{EDM}}}{||\mathbf{x}_{\text{CAM}} - (\mathbf{z}_{\text{CAM}}^T \mathbf{z}_{\text{EDM}})\mathbf{z}_{\text{EDM}}||} \tag{3.42}$$

The points are then converted to an intermediate coordinate frame with $\mathbf{d}_1$ as origin

$$\begin{bmatrix} \hat{\mathbf{d}}_1 & \hat{\mathbf{d}}_2 & \hat{\mathbf{d}}_3 \end{bmatrix}_{[4x3]} = \mathbf{M}_{1,M}^{-1} \mathbf{M}_B^{-1} \begin{bmatrix} \mathbf{d}_1 & \mathbf{d}_1 + \mathbf{x}_{\text{EDM}} & \mathbf{d}_1 + \mathbf{z}_{\text{EDM}} \\ 1 & 1 & 1 \end{bmatrix}_{[4x3]} \tag{3.43}$$

with $M = 3$. The rigid transformation $\mathbf{M}_T^{(\text{EDM})}$ can be extracted according to Sec. 3.1.6, where we define the two ordered point sets as

$$S_a = \left( \mathbf{c}_t, \mathbf{c}_t + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \mathbf{c}_t + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right) \qquad S_b = \left( \tilde{\mathbf{d}}_1, \tilde{\mathbf{d}}_2, \tilde{\mathbf{d}}_3 \right) \tag{3.44}$$

where $\tilde{\mathbf{d}}_i$ is the Euclidean representation of the homogeneous coordinate $\hat{\mathbf{d}}_i$, and $\mathbf{c}_t$ is given by $\begin{bmatrix} 0 & 0 & d_1 \end{bmatrix}^T$.

## 3.2 Model Error Estimation

The DH model error can be expressed using the average point distance $\bar{d}$ and the unbiased standard deviation $\hat{\sigma}$ between recorded and calculated point sets according to

$$\bar{d} = \frac{\sum_{i=1}^N ||\mathbf{x}_{i,\text{meas}} - \mathbf{x}_{i,\text{calc}}||}{N} \qquad \hat{\sigma} = \sqrt{\frac{\sum_{i=1}^N (||\mathbf{x}_{i,\text{meas}} - \mathbf{x}_{i,\text{calc}}|| - \bar{d})^2}{N-1}} \tag{3.45}$$

where $\mathbf{x}_{i,\text{meas}}$ are points of the measurement set and $\mathbf{x}_{i,\text{calc}}$ are points of the calculated point set. The measurement sets for the EDM tool and for the camera tool can be recorded using Eqn. 3.5 and Eqn. 3.6, respectively. The calculation set for the EDM tool is given by Eqn. 3.38. Figure 3.11 shows the basic setup for the modeling error estimation used in this work.

The calculation set for the camera tools can be derived from Eqn. 3.29. A back-projected homogeneous image coordinate $\hat{\mathbf{u}}$ defines a view ray $X_T(\hat{\mathbf{u}}, \mu)$ in the reference frame [44]. The projection of points from the reference frame to the camera image is given by the $3 \times 4$ projection matrix $\mathbf{P}$ according to

$$\mathbf{P} = \mathbf{P}_{\text{CAM}} \mathbf{M}_{B,T}^{-1} = [\mathbf{M}|\mathbf{p}_4] \, . \tag{3.46}$$

The decomposition of the projection matrix $\mathbf{P}$ into the $3 \times 3$ matrix $\mathbf{M}$ and the $3 \times 1$

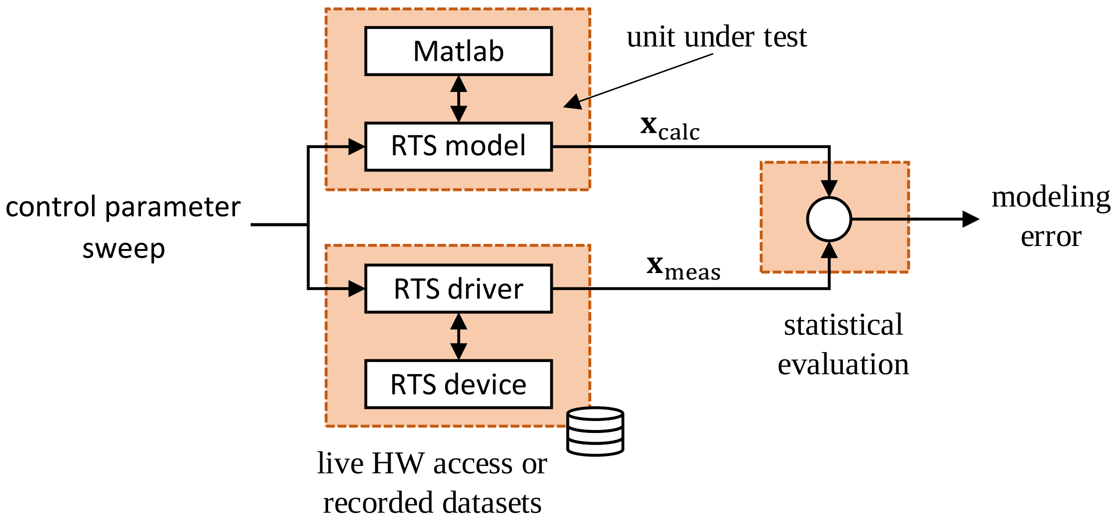


Figure 3.11: Modeling error estimation. In this work, we evaluate the discrepancy between the estimated kinematic model and the geometric model used by the device driver of the exemplary RTS. Note that this does not require any external measurement target, but also does not provide any measurement uncertainty information of the physical device.

column vector $\mathbf{p}_4$ is used for view ray calculation of finite cameras: A numerical robust back-projection of a image coordinate $\hat{\mathbf{u}}$ to the view ray $X_T(\hat{\mathbf{u}}, \mu)$ with metric parametrization of the ray length $\mu$ is given by Klug et al. [66]:

$$X_T(\hat{\mathbf{u}}, \mu) = \frac{\mu}{||\mathbf{M}^{-1} \cdot \hat{\mathbf{u}}||} \begin{bmatrix} \mathbf{M}^{-1} \cdot \hat{\mathbf{u}} \\ 0 \end{bmatrix} + \begin{bmatrix} -\mathbf{M}^{-1} \cdot \mathbf{p}_4 \\ 1 \end{bmatrix} \tag{3.47}$$

Eqn. 3.47 can be used to calculate the evaluation point set for the camera tool frame, using the ray length $\mu = d_{\text{CAM}}$. The reader may refer to the book by Hartley and Zisserman [44] for details about camera and projection matrix properties and alternative projection methods.

Different aspects of the modeling error and the influence of the individual end-effectors can be analyzed. Traditionally, the camera of the RTS is used by the operator to measure a certain 3D point, but the actual measurement does not use image-based measurement (IBM) methods. Scanning applications may not use the camera at all. Ideally, the $\mathbf{z}$ axes of the camera and the EDM are aligned, and the tool rotations around the principal ray and the EDM ray do not influence the result. For such applications, the tool rotations around their respective $\mathbf{z}$-axis should be excluded in the error analysis (method A)[8]. Advanced

[8]Previous work in DH modeling does not address the complete tool pose, but only the position of the end-effector [10, 22, 45, 84, 88].

Table 3.1: Model evaluation taxonomy, showing the most suitable choice for different applications.

| evaluation aspect | variant | usage |
|---|---|---|
| end-effector type | EDM only | non-vision based measurements |
| | camera only | vision-based measurements |
| | both (EDM, camera) | |
| end-effector error | position only (A) | non-vision based measurements |
| | position and rotation around principal ray (B) | vision-based measurements |
| optimization | none | DH modeling workflow evaluation, start values for optimization, comparison with previous work |
| | reduced angle parameter space | evaluate influence of parameter space coverage |
| | complete angle parameter space | |

applications use IBM methods; hence, errors of all tool poses have to be considered (method B). The different aspects of the evaluation are shown in Table 3.1.

## 3.3 Inverse Kinematic Modeling

Given a 3D end-effector pose, the joint parameters can be calculated using the *inverse kinematic model*. However, no general method for calculating a closed-form inverse kinematic model exists [7]. In general, estimating the inverse kinematic model of an open chain requires solving a system with ambiguities, singularities and nonlinearities. Figure 3.12 shows the four different categories of inverse kinematic solvers as proposed by El-Sherbiny et al. [33]. Aristidou et al. [7] provides a more complete survey of inverse kinematic techniques, with a similar categorization.

For special geometric constellations, closed-form inverse kinematic models can be found by means of *geometrical or algebraic approaches*. Closed-form solutions are preferred for reasons of performance and ease-of-implementation, but usually require model restrictions and simplifications. For instance, an RTS can be reduced to a generalized spherical chain as shown in Fig. 3.13. Here, modeling errors are compensated with physical device calibration and with special measurement methods [106]. A closed-form inverse kinematic model can then be derived as described by González-Palacios et al. [40], where the authors present closed-form solutions for different spherical architectures. Note that modeling and calibration methods presented in literature for RTS, such as discussed in the book by Uren and Price [106], or in the publication by Ehrhart [31], are based on such simplified

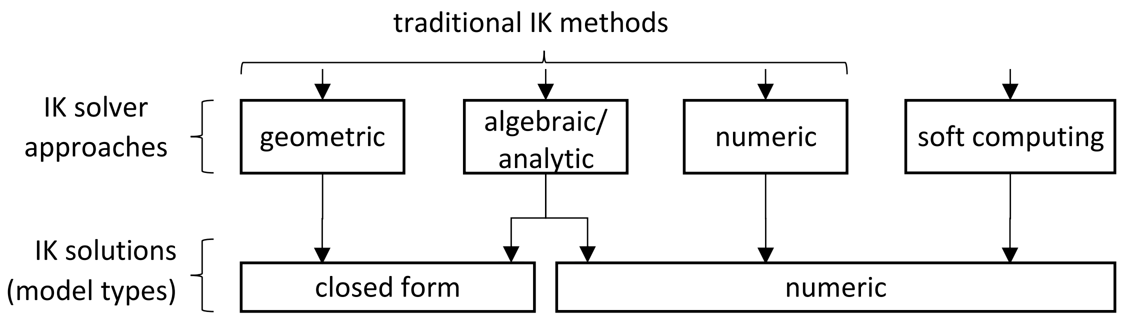


Figure 3.12: Classification of inverse kinematic methods as proposed by El-Sherbiny et al. [33].

spherical chain models.

*Numerical inverse kinematic solvers* are used when no analytical mapping can be found. However, they do not result in a closed-form solution.

In addition to the traditional methods, El-Sherbiny et al. [33] further discuss inverse kinematic solvers based on *soft computing*. Here, solutions for hard problems are approximated using fuzzy logic, evolutionary computation or learning theory.

In this work, we use various model simplifications and derive the inverse kinematic equations from a closed-form spherical model of the exemplary RTS as discussed in Sec. 4.1. Other methods are beyond the scope of this work, but can be found in [7, 33, 34, 40, 58, 75, 110].

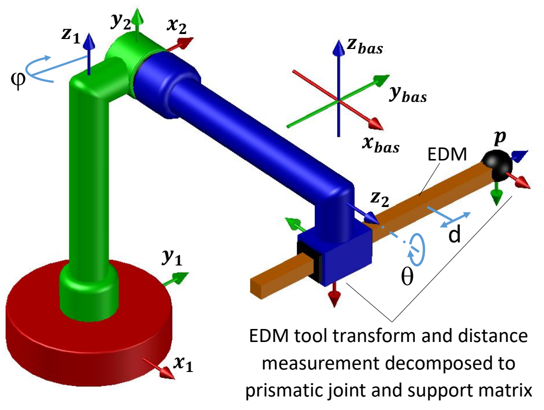
## 3.4 Consistent Model for Kinematics and Point Transfer Functions

The transformation of 3D points between sensor data, base frame and end-effectors can be defined with respect to the forward kinematic model. This provides a generalized and consistent RTS system description and allows better device abstraction for algorithm design, implementation and documentation.

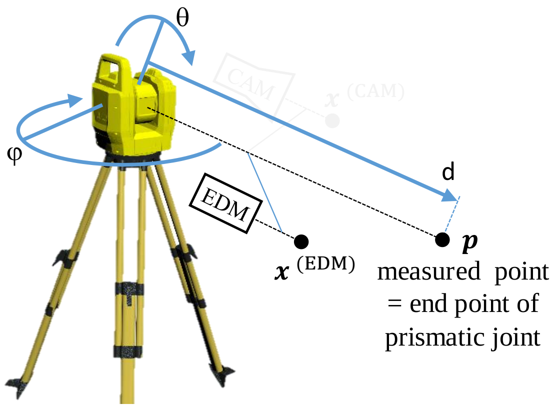
Figure 3.14 shows the relationship between point transformations and coordinate frames with respect to the forward kinematic model. Table 3.2 provides and overview of the required transformations and functions.

## 3.5 Discussion

The modeling approach we proposed in this chapter leads to a generalized and consistent description for RTS position kinematics and 3D point transformations. The concept proved to be useful during our work for estimating a geometric model of an instrument with certain unknown specifications as well as for validating device documentations and for generating


$z_1$
$y_2$
$x_2$
φ
$z_{bas}$
$y_{bas}$
$x_{bas}$
EDM
$p$
$z_2$
d
θ
$y_1$
$x_1$
EDM tool transform and distance measurement decomposed to prismatic joint and support matrix

(a) RRP chain



(b) RTS distance measurement model

Figure 3.13: Generalized spherical architecture as described by González-Palacios *et al.* [40]. (a) RTS distance measurements described by an RRP chain, where two revolute joints at the base describe the pan-tilt unit, and a prismatic joint describes the EDM. (b) RTS distance measurement.

various different simulation setups. Once implemented, the effort for using this concept for different devices and measurement setups was minimal.

While the general forward position kinematic model that we estimate in this chapter is sufficient for many applications, inverse kinematics has not been addressed to a great extend. However, as no general solution for the inverse kinematic problem of spatial linkages exists, we postpone this description to the following chapter, where we model an exemplary RTS.

In the following chapter, we apply the model extraction method to an exemplary RTS and discuss critical simulation properties when the extracted model is used together with 3D rendering engines.

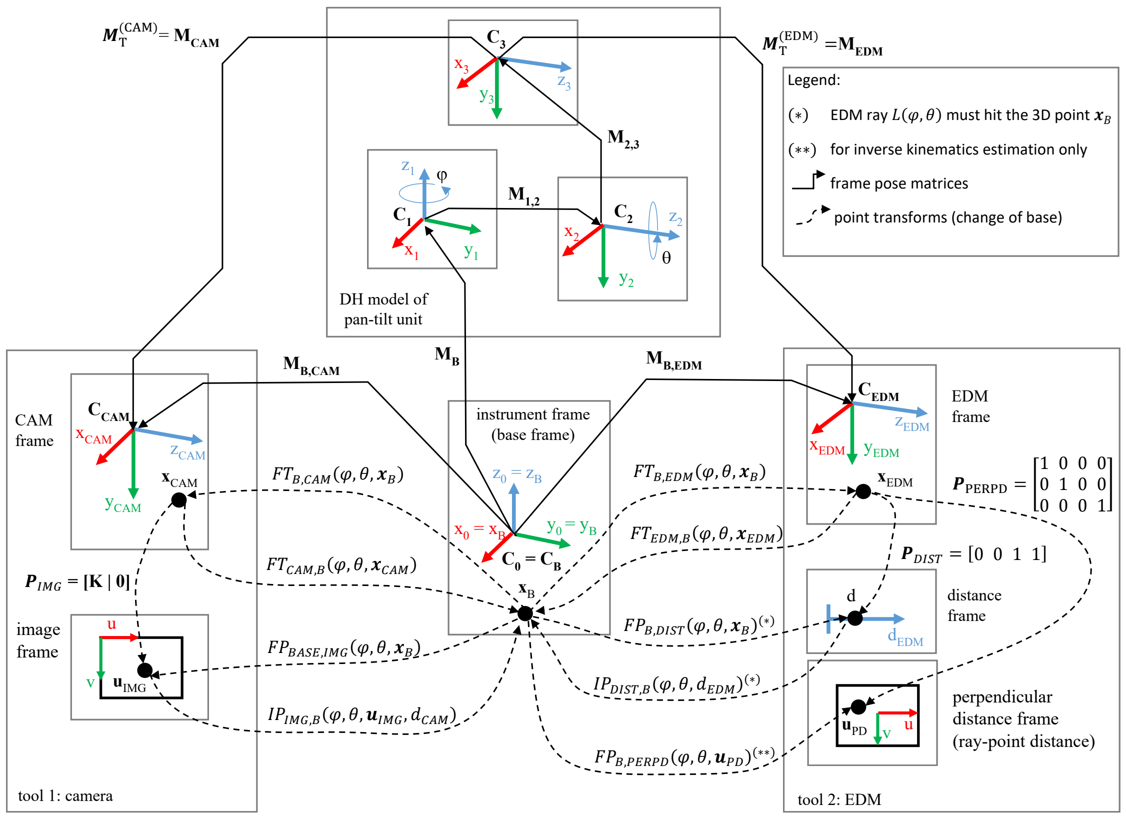

$M_T^{(CAM)} = \mathbf{M}_{CAM}$
$M_T^{(EDM)} = \mathbf{M}_{EDM}$
$C_3$
$x_3$
$y_3$
$z_3$
$\mathbf{M}_{2,3}$
$z_1$
$\varphi$
$C_1$
$x_1$
$y_1$
$\mathbf{M}_{1,2}$
$C_2$
$x_2$
$y_2$
$z_2$
$\theta$
DH model of pan-tilt unit
Legend:
(∗) EDM ray $L(\varphi,\theta)$ must hit the 3D point $\boldsymbol{x}_B$
(∗∗) for inverse kinematics estimation only
frame pose matrices
point transforms (change of base)
$\mathbf{M}_B$
$\mathbf{M}_{B,CAM}$
$\mathbf{M}_{B,EDM}$
CAM frame
$C_{CAM}$
$x_{CAM}$
$y_{CAM}$
$z_{CAM}$
$\mathbf{x}_{CAM}$
$P_{IMG} = [\mathbf{K} \mid \mathbf{0}]$
image frame
u
v
$\mathbf{u}_{IMG}$
tool 1: camera
instrument frame (base frame)
$z_0 = z_B$
$x_0 = x_B$
$y_0 = y_B$
$C_0 = C_B$
$\mathbf{x}_B$
$FT_{B,CAM}(\varphi,\theta,\boldsymbol{x}_B)$
$FT_{CAM,B}(\varphi,\theta,\boldsymbol{x}_{CAM})$
$FP_{BASE,IMG}(\varphi,\theta,\boldsymbol{x}_B)$
$IP_{IMG,B}(\varphi,\theta,\boldsymbol{u}_{IMG},d_{CAM})$
$FT_{B,EDM}(\varphi,\theta,\boldsymbol{x}_B)$
$FT_{EDM,B}(\varphi,\theta,\boldsymbol{x}_{EDM})$
$FP_{B,DIST}(\varphi,\theta,\boldsymbol{x}_B)^{(*)}$
$IP_{DIST,B}(\varphi,\theta,d_{EDM})^{(*)}$
$FP_{B,PERPD}(\varphi,\theta,\boldsymbol{u}_{PD})^{(**)}$
$C_{EDM}$
EDM frame
$x_{EDM}$
$y_{EDM}$
$z_{EDM}$
$\mathbf{x}_{EDM}$
$P_{PERPD} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
$P_{DIST} = [0\ 0\ 1\ 1]$
d
$d_{EDM}$
distance frame
$\mathbf{u}_{PD}$
u
v
perpendicular distance frame (ray-point distance)
tool 2: EDM

Figure 3.14: RTS point transformations according to forward kinematic model.

Table 3.2: Generalized point transfer functions, forward and inverse kinematics of an RTS with one camera and one EDM.

| | sensor | function/matrix | space transform (Cartesian representation) |
|---|---|---|---|
| forward kinematics | | forward sensor projection matrices (homogeneous coordinates) | |
| | CAM | $\mathbf{P}_{IMG} = \begin{bmatrix} f_x & 0 & d_x & 0 \\ 0 & f_y & d_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$ Note that the projections might give you a negative value for the depth or distance (reflection). | 3D $\mapsto$ 2D |
| | EDM | $\mathbf{P}_{DIST} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ | 3D $\mapsto$ 1D |
| | | $\mathbf{P}_{PERPD} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ | 2D $\mapsto$ 2D |
| | | forward kinematic frame pose matrices (homogeneous coordinates) | |
| | CAM | $\mathbf{M}_{B,T}^{(CAM)} = \mathbf{M}_{B,CAM}$ | 3D $\mapsto$ 3D |
| | EDM | $\mathbf{M}_{B,T}^{(EDM)} = \mathbf{M}_{B,EDM}$ | 3D $\mapsto$ 3D |
| | | forward kinematic point transforms (homogeneous coordinates) | |
| | CAM | $\hat{\mathbf{x}}_B = FT_{B,CAM}(\varphi, \theta, \hat{\mathbf{x}}_{CAM}) = \mathbf{M}_{B,CAM}(\varphi, \theta) \cdot \hat{\mathbf{x}}_{CAM}$ | 3D $\mapsto$ 3D |
| | | $\hat{\mathbf{x}}_{CAM} = FT_{B,CAM}(\varphi, \theta, \hat{\mathbf{x}}_B)^{-1} = \mathbf{M}_{B,CAM}(\varphi, \theta)^{-1} \cdot \hat{\mathbf{x}}_B$ | 3D $\mapsto$ 3D |
| | EDM | $\hat{\mathbf{x}}_B = FT_{B,EDM}(\varphi, \theta, \hat{\mathbf{x}}_{EDM}) = \mathbf{M}_{B,EDM}(\varphi, \theta) \cdot \hat{\mathbf{x}}_{EDM}$ | 3D $\mapsto$ 3D |
| | | $\hat{\mathbf{x}}_{EDM} = FT_{B,EDM}(\varphi, \theta, \hat{\mathbf{x}}_B)^{-1} = \mathbf{M}_{B,EDM}(\varphi, \theta)^{-1} \cdot \hat{\mathbf{x}}_B$ | 3D $\mapsto$ 3D |
| | | forward point projections (homogeneous coordinates) | |
| | CAM | $\hat{\mathbf{u}} = FP_{B,IMG}(\varphi, \theta, \hat{\mathbf{x}}_B) = P_{IMG} \cdot \mathbf{M}_{B,CAM}(\varphi, \theta)^{-1} \cdot \hat{\mathbf{x}}_B$ | 3D $\mapsto$ 2D |
| | EDM | $\hat{d} = FP_{B,DIST}(\varphi, \theta, \hat{\mathbf{x}}_B) = P_{DIST} \cdot \mathbf{M}_{B,EDM}(\varphi, \theta)^{-1} \cdot \hat{\mathbf{x}}_B$ | 3D $\mapsto$ 1D if ray hits target |
| | | $\hat{\mathbf{u}} = FP_{B,PERPD}(\varphi, \theta, \hat{\mathbf{x}}_B) = P_{PERPD} \cdot \mathbf{M}_{B,EDM}(\varphi, \theta)^{-1} \cdot \hat{\mathbf{x}}_B$ | 3D $\mapsto$ 2D (this is NOT the image space) |
| | | inverse sensor projection matrices (homogeneous coordinates) | |
| | CAM | $\hat{\mathbf{x}}_B = IP_{IMG,B}(\varphi, \theta, \hat{\mathbf{u}}, d) = \mathbf{M}_{B,CAM}(\varphi, \theta)^{-1} \cdot IP_{IMG,CAM}(\hat{\mathbf{u}}, d)$ | $(2D, depth) \mapsto$ 3D |
| | | $\hat{\mathbf{x}}_{CAM} = IP_{IMG,CAM}(\hat{\mathbf{u}}, d) = X_T(\hat{\mathbf{u}}, d)$ | $(2D, depth) \mapsto$ 3D |
| | EDM | $\hat{\mathbf{x}}_B = IP_{DIST,B}(\varphi, \theta, d) = \mathbf{M}_{B,EDM}(\varphi, \theta)^{-1} \cdot IP_{DIST,EDM}(d)$ | 1D $\mapsto$ 3D |
| | | $\hat{\mathbf{x}}_{EDM} = IP_{DIST,EDM}(d) = \begin{bmatrix} 0 & 0 & d & 1 \end{bmatrix}^T$ | 1D $\mapsto$ 3D |
| inverse kinematics | | inverse kinematic point transforms (Cartesian coordinates) | |
| | CAM | $(\varphi, \theta) = IK_{CAM}(\mathbf{x}_B, \mathbf{u}) = \min \lvert\lvert HC(FP_{B,IMG}(\varphi, \theta, \hat{\mathbf{x}}_B)) - \mathbf{u} \rvert\rvert$ | 2D $\mapsto$ DH control parameters |
| | | $(\varphi, \theta) = IK_{CAM}(\mathbf{x}_B) = IK_{CAM}(\hat{\mathbf{x}}_B, \hat{\mathbf{u}}_c)$, with principal point offset $\mathbf{u}_c$ | 3D $\mapsto$ DH control parameters |
| | EDM | $(\varphi, \theta) = IK_{EDM}(\mathbf{x}_B) = \min \lvert\lvert HC(IP_{DIST,B}(\varphi, \theta, d)) - \mathbf{x}_B \rvert\rvert$ with $d = \lvert\lvert HC(FT_{B,EDM}(\varphi, \theta, \hat{\mathbf{0}})) - \mathbf{x}_B \rvert\rvert$ | 3D $\mapsto$ DH control parameters |
| | | $(\varphi, \theta) = IK_{EDM}(\mathbf{x}_B) = \min \lvert\lvert HC(FP_{B,PERPD}(\varphi, \theta, \mathbf{x}_B)) - \mathbf{0} \rvert\rvert$ | 3D $\mapsto$ DH control parameters |

# 4 Kinematic Modeling and Simulation of the Exemplary RTS

In this chapter, we apply the modeling method of chapter 3 to the exemplary RTS. To cover a wide spectrum of possible applications, we compare the DH results to the related spherical model, which represents simplified and idealized relationships between actuators and sensors, and the numerically optimized model of the exemplary RTS.

## 4.1 Forward Kinematic Model

The data set for the DH model extraction of the device contains 15 points for each joint, recorded with the API functions given in Eqn. 3.5 and Eqn. 3.6. Endpoint positions for the first joint were recorded by setting the control variable $\theta$ to the fixed value $\theta = \frac{\pi}{2}$ while varying $\varphi$ from 0 to $2\pi$. Endpoint positions for the second joint were recorded by setting the control variable $\varphi$ to the fixed value $\varphi = 0$ while varying $\theta$ from 0 to $\pi$. The angular parameter space coverage of the data set is shown in Fig. 3.6b. For each position, the length of the back projected EDM ray and the length of the back-projected camera view ray were set to one meter ($d_1 = 1$). The pixel position for the back-projected camera view ray was aligned with the principal point of the camera, which was also provided by the API of the device. For EDM tool pose extraction, a second ray at each control position was extracted using a constant ray length of two meter ($d_2 = 2$). For camera tool pose extraction, two additional rays at each position were used, for which the back-projected image coordinates were shifted by one pixel in $\mathbf{x}$ and one pixel in $\mathbf{y}$ direction, respectively.

Table 4.1 shows the extracted DH parameters, Table 4.2 shows the base and tool transforms for both, the EDM and camera end-effectors. The complete extracted geometric model, including all coordinate frames and transformation matrices, is shown in Figure 3.2b.

Table 4.1: Estimated DH parameters for the exemplary RTS

| joint | $\alpha_i$ | $a_i$ | $\gamma_i$ | $d_i$ |
|---|---|---|---|---|
| 1 | 1.5708 | 0.0 | $\varphi_i - 1.5708$ | 0.0 |
| 2 | 0.0 | 0.0 | $\theta_i$ | 0.0 |

Table 4.2: Estimated base and end-effector transforms for the exemplary RTS

| transform | $\alpha$ | $\beta$ | $\gamma$ | $t_x$ | $t_y$ | $t_z$ |
|---|---|---|---|---|---|---|
| $\mathbf{M}_B$ | 3.1416 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| $\mathbf{M}_T^{(\mathrm{CAM})}$ | -1.5020 | 1.5646 | 3.2103 | -0.0003 | 0.0 | 0.0003 |
| $\mathbf{M}_T^{(\mathrm{EDM})}$ | -1.5020 | 1.5646 | 3.2103 | -0.0003 | 0.0 | 0.0003 |

## 4.2 Model Simplification

The extracted model parameters in Sec. 4.1 are sufficient for building a device simulator. However, an idealized device simulation is often preferred. Idealized models can be used for analytic system and algorithm design and verification, for calculating or simulating desired system behaviors or for generating reference data sets.

The device used for this work shows similarities in the tool transforms, translation parameters, which are close to zero, and rotation parameters, which can be approximated by multiples of $\frac{\pi}{2}$. In this section, we apply numerical approximations to relate the extracted geometric model with the simplified and idealized spherical model. By setting the translation parameters to 0 and replacing the rotational components by the nearest multiple of $\frac{\pi}{2}$, we get following forward kinematic model of the exemplary RTS:

$$\left.\begin{array}{c} \mathbf{M}_{B,T} = \mathbf{M}_B \prod_{n=1}^{2} \mathbf{M}_{n,n+1} \mathbf{M}_T \qquad \mathbf{M}_B = \mathbf{R}_x(\pi) \qquad \mathbf{M}_{1,2} = \mathbf{R}_z(\varphi_i - \frac{\pi}{2}) \mathbf{R}_x(\frac{\pi}{2}) \\ \mathbf{M}_{2,3} = \mathbf{R}_z(\theta_i) \qquad \mathbf{M}_T = \mathbf{R}_z(\pi)\mathbf{R}_y(\frac{\pi}{2})\mathbf{R}_x(-\frac{\pi}{2}) \end{array}\right\} \tag{4.1}$$

The rotation matrices $\mathbf{R}_x$, $\mathbf{R}_y$ and $\mathbf{R}_z$ are defined in Eqn. A.4.
Multiplying and simplifying matrix $\mathbf{M}_{B,T}$ leads to

$$\mathbf{M}_{B,T} = \begin{bmatrix} \cos(\varphi_i) & \cos(\theta_i)\sin(\varphi_i) & \sin(\varphi_i)\sin(\theta_i) & 0 \\ -\sin(\varphi_i) & \cos(\varphi_i)\cos(\theta_i) & \cos(\varphi_i)\sin(\theta_i) & 0 \\ 0 & -\sin(\theta_i) & \cos(\theta_i) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

which can be written as an Euler rotation according to Eqn. A.5:

$$\mathbf{M}_{B,T} = \mathbf{R}_z(-\varphi_i)\mathbf{R}_y(0)\mathbf{R}_x(-\theta_i) \tag{4.3}$$

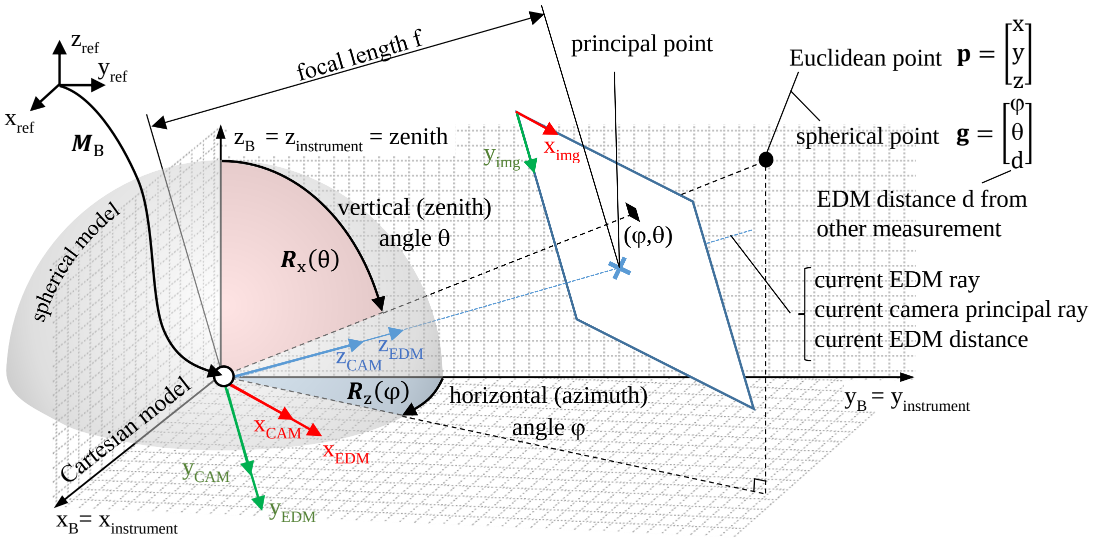


Figure 4.1: Simplified geometric model for a calibrated RTS [65] as used by the exemplary RTS for this work. Azimuth angle, zenith angle and radial distance $d$ are denoted by $\varphi$, $\theta$, and $d$, respectively. In this simplified version, the coordinate system of the EDM is aligned with the camera coordinate system as well as the spherical coordinate frame of the RTS. The 4x4 transformation matrix $\mathbf{M}_B$ is a six DOF describes the RTS pose with respect to a common reference frame.

## 4.3 Simplified Forward Kinematic Model

The model simplification given in Sec. 4.2 leads to a kinematic model for the exemplary RTS with aligned camera tool frame and EDM tool frame $\mathbf{M}_T^{(CAM)} = \mathbf{M}_T^{(EDM)} = \mathbf{M}_T$.

Let $\mathbf{g} = [\varphi \quad \theta \quad d]^T$ be a spherical coordinate vector with vector length $d$. Then, the vector $\mathbf{g}$ can be interpreted as tuple of angle control parameter $(\varphi, \theta)$ and the measured EDM distance $d$. The vector $\mathbf{g}$ can also be interpreted as back-projected image pixel ray, given as spherical coordinates. The conversion of image coordinates to spherical rays allows for a common forward and inverse kinematic description for both, the EDM and the camera tool. Furthermore, this allows rotation-invariant selection of targets in the image as shown in Figure 4.1. For EDM measurements, $d$ is the measured distance, for image data, $d$ is a depth information for the related pixel, determined by previous EDM measurements of from external sources[1].

The operator $G : \mathbf{g} \rightarrow \mathbf{p}$ that maps the spherical vector $\mathbf{g}$ to an Euclidean 3D point

[1]The exemplary device does not have a depth camera.

$\mathbf{p} = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}^T$ is then given by:

$$G(\varphi, \theta, d) := \mathbf{Q}(\varphi, \theta) \begin{bmatrix} 0 \\ 0 \\ d \\ 1 \end{bmatrix} \qquad \hat{\mathbf{p}} = \begin{bmatrix} d\sin(\varphi)\sin(\theta) \\ d\cos(\varphi)\sin(\theta) \\ d\cos(\theta) \\ 1 \end{bmatrix} \tag{4.4}$$

where $\hat{\mathbf{p}}$ is the homogeneous representation of the Euclidean 3D point $\mathbf{p}$. The combined rotation matrix $\mathbf{Q}(\varphi, \theta)$ is defined by

$$\mathbf{Q}(\varphi, \theta) = \mathbf{R}_z(-\varphi)\mathbf{R}_y(0)\mathbf{R}_x(-\theta) = \mathbf{M}_{B,T} \tag{4.5}$$

with the $4x4$ transformation matrix $\mathbf{M}_{B,T}$ given in Eqn. 4.2. The conversion of an Euclidean 3D point in camera tool frame to an image pixel is given by Eqn. 3.29. This means that the forward kinematic model as shown in Fig. 3.14 is fully defined by Eqn. 4.4, Eqn. 4.5, and the projection matrices $\mathbf{P}_{IMG}$ and $\mathbf{P}_{DIST}$.

## 4.4 Simplified Inverse Kinematic Model

Let the operator $F : \mathbf{p} \to \mathbf{g}$ define the inverse kinematic model that maps the Euclidean 3D point $\mathbf{p}$ to the spherical point $\mathbf{g}$. Then, the inverse kinematic model can be derived directly from Eqn. 4.4: The ray length $d$ is given by the Euclidean vector length $d = \sqrt{p_x^2 + p_y^2 + p_z^2}$. The vertical angle $\theta$ can be calculated by

$$p_z = d\cos(\theta) \Rightarrow \theta = \arccos(\frac{p_z}{d}) \tag{4.6}$$

The horizontal angle $\varphi$ can be calculated by

$$\frac{p_x}{p_y} = \frac{d \sin\varphi \sin\theta}{d\cos\varphi\sin\theta} \Rightarrow \varphi = \arctan(\frac{p_x}{p_y}). \tag{4.7}$$

However, we use an alternative formulation, which favors the trigonometric function *atan*2 over the trigonometric functions *arccos* and *arctan* for numerical stability reasons. Hence, the complete inverse kinematic model can be written as

$$F(\mathbf{p}) := \begin{bmatrix} arccos\left(\frac{p_z}{\sqrt{p_x^2+p_y^2+p_z^2}}\right) = atan2(\sqrt{p_x^2+p_y^2}, p_z) \\ arctan\left(\frac{p_x}{p_y}\right) = atan2(p_x, p_y) \\ \sqrt{p_x^2+p_y^2+p_z^2} \end{bmatrix} = \begin{bmatrix} \varphi \\ \theta \\ d \end{bmatrix} \tag{4.8}$$

Table 4.3: Kinematic model error of the extracted DH model, $e_{DH}$, and of the simplified model, $e_{simplified}$. The individual and compound errors for the end-effectors are provided. (A) Tool rotations around principal ray and EDM ray ignored. (B) Tool rotations around principal ray and EDM ray considered

| method | end-effector | $e_{\mathrm{DH}}$ $[m]$ | $e_{\mathrm{simplified}}$ $[m]$ |
|---|---|---|---|
| A | EDM | $8.16\times10^{-04} \pm 1.86\times10^{-16}$ | $3.98\times10^{-17} \pm 4.91\times10^{-17}$ |
| | camera | $2.97\times10^{-06} \pm 9.47\times10^{-07}$ | $8.14\times10^{-04} \pm 1.02\times10^{-06}$ |
| | compound | $4.10\times10^{-04} \pm 4.07\times10^{-04}$ | $4.07\times10^{-04} \pm 4.07\times10^{-04}$ |
| B | EDM | $1.04\times10^{-03} \pm 2.20\times10^{-04}$ | $7.40\times10^{-17} \pm 9.60\times10^{-17}$ |
| | camera | $1.66\times10^{-04} \pm 1.15\times10^{-04}$ | $4.38\times10^{-03} \pm 2.52\times10^{-03}$ |
| | compound | $5.14\times10^{-04} \pm 4.57\times10^{-04}$ | $2.63\times10^{-03} \pm 2.90\times10^{-03}$ |

with

$$-\pi \leq \varphi < \pi \qquad 0 \leq \theta < \pi \qquad d > 0 \tag{4.9}$$

This means that the inverse kinematic operations of the simplified model given in Fig. 4.1 is fully defined by Eqn. 4.8, Eqn. 4.9 and the pseudo-inverse of the projection matrices, $\mathbf{P}^{\dagger}_{IMG}$ and $\mathbf{P}^{\dagger}_{DIST}$, which unproject measured sensor data to the 3D space. Note that the ray equation given in Eqn. 3.47 provides numeric stable and scale-preserving alternative to $\mathbf{P}^{\dagger}_{IMG}$ for converting a image pixel $\mathbf{u}$ with known pixel depth to a 3D point.

## 4.5 Model Error Estimation

The modeling error of the exemplary RTS as given in Sec. 3.2 describes the discrepancy between the kinematic model and the model used by the device driver. Table 4.3 shows the result of the error analysis for the DH geometric model and the simplified geometric model. The results show that the simplified, spherical model, compared to the geometrically extracted DH model, is a good approximation for the EDM end-effector; the DH model shows lower modeling error for the complete system.

## 4.6 Model Optimization

The modeling method proposed in Sec. 3.1 is a greedy algorithm, which only optimizes local cost functions. While this is sufficient for many applications, the estimated model can be refined using nonlinear optimization techniques to decrease the error of the model. In this section, we briefly discuss model optimization using a global error function. However, this is only a proof of concept, while a detailed optimization analysis is beyond the scope of this work.

Table 4.4: Errors of the optimized model, with tool rotations around principal ray and EDM ray considered (method B)

| end-effector | $e_{\text{optimized}}$ $[m]$ | $e_{\text{optimized,cross}}$ $[m]$ | $e_{\text{optimized,full}}$ $[m]$ |
|---|---|---|---|
| EDM | $1.72{\times}10^{-5} \pm 5.19{\times}10^{-6}$ | $2.12{\times}10^{-05} \pm 7.64{\times}10^{-06}$ | $1.99{\times}10^{-05} \pm 6.88{\times}10^{-06}$ |
| camera | $1.98{\times}10^{-5} \pm 8.20{\times}10^{-6}$ | $2.16{\times}10^{-05} \pm 9.50{\times}10^{-06}$ | $1.41{\times}10^{-05} \pm 7.79{\times}10^{-06}$ |
| compound | $1.87{\times}10^{-5} \pm 7.26{\times}10^{-6}$ | $2.15{\times}10^{-05} \pm 8.81{\times}10^{-06}$ | $1.64{\times}10^{-05} \pm 7.95{\times}10^{-06}$ |

Table 4.5: DH parameter results of the sequential quadratic programming (SQP) optimization

| i | $\alpha_i$ | $a_i$ | $\gamma_i$ | $d_i$ |
|---|---|---|---|---|
| 1 | 1.570808 | 0.0 | $\varphi_i - 1.5203$ | 0.0 |
| 2 | $0.2055{\times}10^{-3}$ | $-0.1117{\times}10^{-3}$ | $\theta_i{-}0.4432{\times}10^{-3}$ | $0.0684{\times}10^{-3}$ |

Table 4.6: Base and tool transformation results of the SQP optimization.

| transform | $\alpha$ | $\beta$ | $\gamma$ | $t_x$ | $t_y$ | $t_z$ |
|---|---|---|---|---|---|---|
| $\mathbf{M}_B$ | 3.1416 | 0.0 | 0.0504 | $5.915{\times}10^{-9}$ | $7.216{\times}10^{-9}$ | $3.312{\times}10^{-9}$ |
| $\mathbf{M}_T^{(\text{CAM})}$ | −1.4632 | 1.5646 | 3.2103 | $-0.1451{\times}10^{-3}$ | $-2.152{\times}10^{-9}$ | $0.1505{\times}10^{-3}$ |
| $\mathbf{M}_T^{(\text{EDM})}$ | −1.4219 | 1.5693 | 3.2910 | $0.1421{\times}10^{-3}$ | $1.415{\times}10^{-6}$ | $-9.014{\times}10^{-5}$ |

Finding an optimal model can be formalized as nonlinear optimization problem with boundary conditions and linear scalarization

$$\min_{S_o}(\omega_d \bar{d}_d + \omega_\sigma \hat{\sigma}_d) \text{ subject to } t_{\min} < t_i < t_{\max} \text{ and } r_{\min} < r_i < r_{\max} \tag{4.10}$$

where $\omega_d$ and $\omega_\sigma$ are weighting factors and where the mean value $\bar{d}_d$ and the unbiased standard deviation $\hat{\sigma}_d$ are given in Eqn. 3.45. The parameter set $S_o$ contains the 26 model parameters[2]:

$$S_o = \left(\mathbf{r}_B, \mathbf{t}_B, Q_1, Q_2, \mathbf{r}_T^{(\text{CAM})}, \mathbf{t}_T^{(\text{CAM})}, \mathbf{r}_T^{(\text{EDM})}, \mathbf{t}_T^{(\text{EDM})}\right) \tag{4.11}$$

The two parameter sets $Q_1$ and $Q_2$ contain the eight DH parameters of the model. The rigid base transform is described by the $3 \times 1$ vector $\mathbf{r}_B$, containing the Euler angles, and the $3 \times 1$ translation vector $\mathbf{t}_B$. Analogously, $\left(\mathbf{r}_T^{(\text{CAM})}, \mathbf{t}_T^{(\text{CAM})}\right)$ and $\left(\mathbf{r}_T^{(\text{EDM})}, \mathbf{t}_T^{(\text{EDM})}\right)$ describe the rigid end-effector transforms. We used the lower boundary $t_{\min} = -1{\times}10^{-2}m$ and the upper boundary $t_{\max} = 1{\times}10^{-2}m$ for all translational model parameters. Additionally, we used the lower boundary $r_{\min} = -\pi$ and the upper boundary $r_{\max} = +\pi$ for all rotational

[2]We do not include the camera intrinsics, EDM calibration parameters or scale factors of the system control parameters $(\varphi, \theta)$.

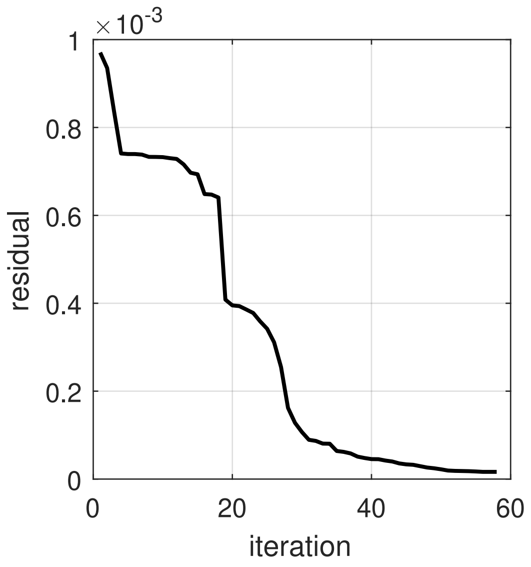


Figure 4.2: Residual evolution of SQP optimization.

model parameters[3].

SQP was used to refine the initial geometric model, which is a gradient-based iterative numerical optimization method. Details about SQP can be found in the book by Nocedal and Wright [80].

The optimization was implemented and executed with MATLAB 2017 and the MATLAB Optimization Toolbox [72], using four parallel sub-processes, Microsoft Windows 10, an Intel Core i7 processor with 64GB RAM. The run-time of the SQP based optimization was $1.326 \times 10^3 s$ ($\approx 0.4h$).

The error results of the optimized model $e_{\text{optimized}}$ are shown in Table 4.4; the optimized model parameters are shown in Table 4.5 and Table 4.6, respectively.

Usually the weights of the objectives are normalized to one, hence $\omega_d + \omega_\sigma = 1$ [30]. In this work, however, we weighted the mean value with $\omega_d = 1.0$, and, the unbiased standard deviation with $\omega_\sigma = 1.0$, for better comparison between the non-optimized result given in Table 4.3 and the evolution of the residual over the optimization iterations shown in Fig. 4.2.

Error distributions with respect the angle control parameter space are shown in Fig. 4.3 and Fig. 4.4, respectively. Note that the high dynamic range of the modeling errors does not allow for a common heat map encoding.

[3]We mapped the result to the range $[0, 2\pi]$ for better comparison between optimized and non-optimized model.

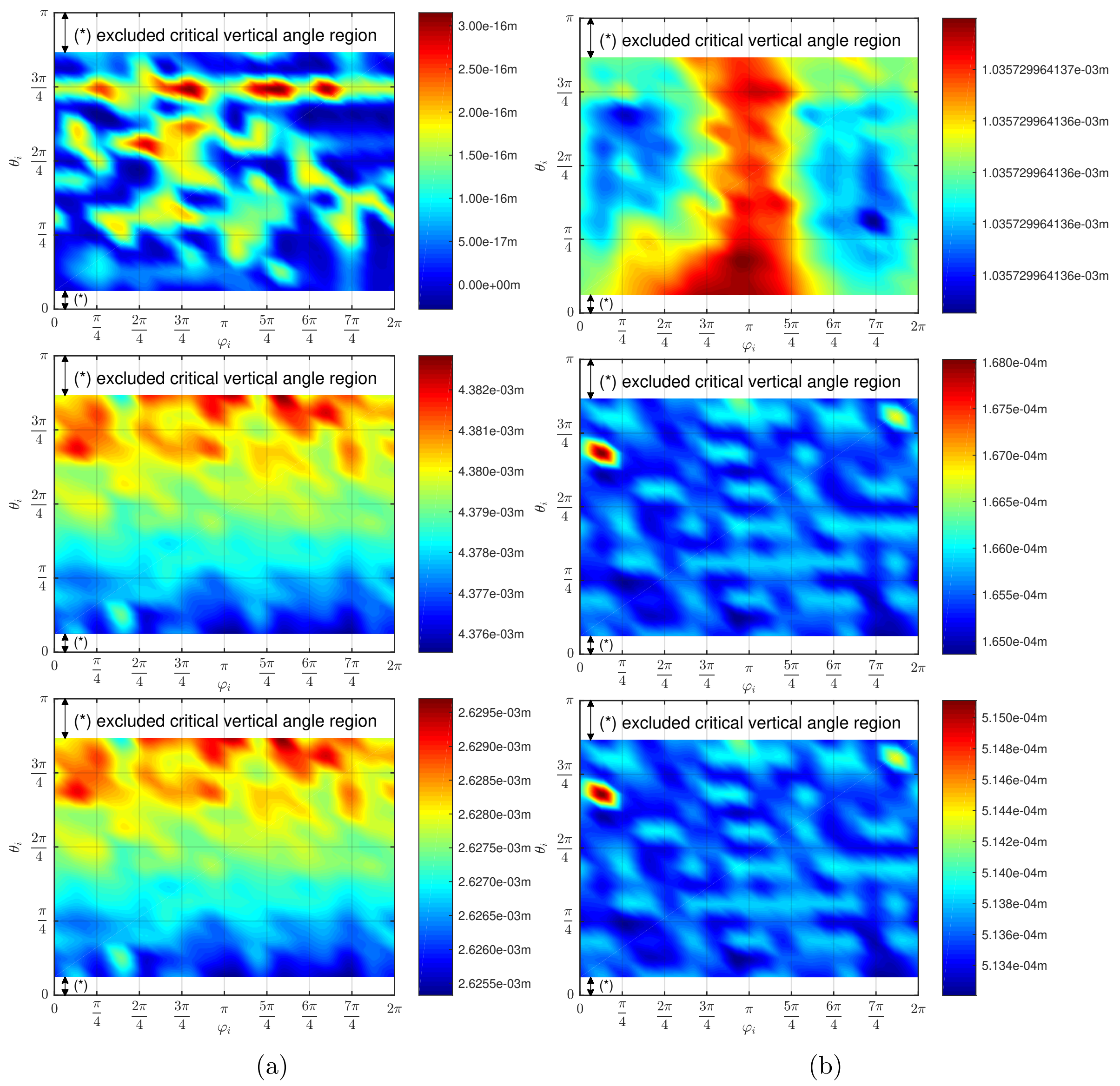


Figure 4.3: Error distributions of estimated DH model with respect to the recorded data set, shown as function of the horizontal and vertical control parameters $(\varphi_i, \theta_i)$, without downstream global optimization scheme. In particular, the pose error of the end-effectors are encoded as colors ( method B.) From left to right column: (a) simplified model, (b) geometrically extracted model. From top to bottom row: EDM, camera, compound.

## 4.7 Interpretation of the Modeling Error

The model error $e_{\text{optimized}}$ of Table 4.4 was calculated with the reduced control parameter range for $\varphi_i$ and $\theta_i$, as shown in Figure 3.6b, for both optimization and evaluation. For simple cross-validation, the optimized model was applied to all samples of the full control

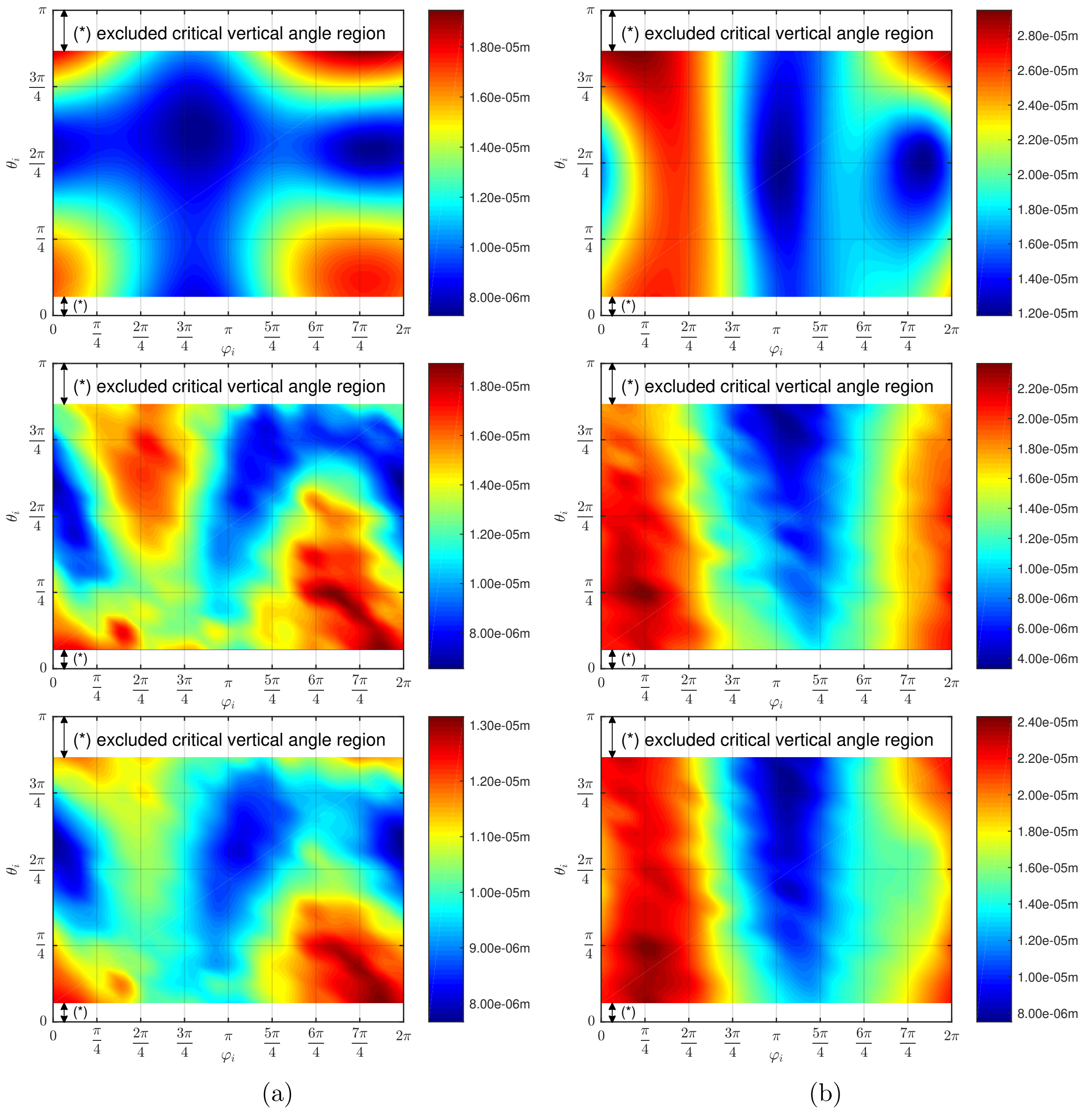


Figure 4.4: Error distributions of estimated DH model with respect to the recorded data set, shown as function of the horizontal and vertical control parameters $(\varphi_i, \theta_i)$, with downstream global optimization scheme. In particular, the pose error of the end-effectors are encoded as colors ( method B.) From left to right column: (a) reduced parameter space used for optimization, (b) full parameter space used for optimization. From top to bottom row: EDM, camera, compound.

parameter space, as shown in Figure 3.6c, which corresponds to the error $e_{\text{optimized,cross}}$ in Table 4.4. Cross-validation is explained in the book by Witten and Frank [109]. In Table 4.4, $e_{optimized,full}$ shows the error of the optimized model, where the full control parameter space for model fitting and validation was used. The errors for all three methods, $e_{\text{optimized}}$,

$e_{\text{optimized,cross}}$ and $e_{\text{optimized,full}}$, were calculated using Eqn. 3.45. The error values of all three methods are in the same order of magnitude, which shows that the sub-set of the recorded samples is sufficient for RTS model optimization. However, a slight decrease in the modeling error can be observed when using samples from the full angle control parameter space[4].

The comparison of Table 4.3 with Table 4.4 shows a decrease in the modeling error by one order of magnitude when applying numerical optimization subsequent to the DH parameter estimation. Hence, numerical optimization is essential for kinmeatic modeling.

Figure 4.3 and Figure 4.4 show the distribution of the modeling error with respect to the angle control parameter space for the non-optimized and the optimized model, respectively. Sample points in the *critical vertical angle regions* were excluded throughout this work for stability reasons. In Figure 4.4, samples of the full control parameter space were used for optimization and validation. The critical regions exclude samples near the poles of the spherical model from the calculations. The poles are defined by $\theta_i = \{0, \pi\}$. In particular, the critical region around $\theta_i = \pi$ also excludes the non-measurable area of a physical RTS, as shown in Figure 1.1. In this work, we set critical regions to $|\theta_i| = \{0, \frac{3}{4}\pi \dots \pi\}$.

All calculations were carried out with 64-bit precision arithmetic [51]. The range of the EDM error distribution of the simplified model, shown in the top left diagram of Figure 4.3, is in the magnitude of the round-off effects of the 64-bit floating-point arithmetic. Hence, the error distribution can be considered as noise, introduced by round-off effects. The error distribution of the geometrically estimated DH model of the EDM end-effector is shown in the top right diagram of Figure 4.3; accuracy and precision are in the magnitude of $1{\times}10^{-3}m$ and $1{\times}10^{-16}m$, respectively. The distribution indicates an EDM pose error of the model, which appears as increasing error between $|\varphi| = \{0 \dots \pi\}$, when applying horizontal rotations. The approximately uniform distribution over the vertical parameter space $\theta = \{0 \dots \pi\}$ indicates that the major offset is along the y-axis of the instrument frame. A scale and offset error of the angle control parameter $\varphi$ could also lead to a similar error distribution, but is contradicted by the error analysis of the simplified EDM model.

For each model in Figure 4.3 and Figure 4.4, the error distributions of the camera end-effector are given in the middle row; the compound error distributions are given in the bottom row.

When comparing all analyzed models, the simplified model is the best match for the device, if only the EDM end-effector is considered. In particular, the EDM modeling error distribution, shown in the top left diagram of Figure 4.3, is in the magnitude of numerical round-off effects. This indicates that the driver uses a spherical coordinate system to convert angle and distance sensor data $(\varphi_i, \theta_i, d_i)$ to Euclidean points $\mathbf{x}_i$.

When only considering the camera end-effector, the accuracy of the geometrically estimated model is by a order of magnitude lower, compared to the simplified model; the error distribution is more uniform with respect to the angle parameter space. The

[4]The analysis of the effect of using a reduced distance parameter space is beyond the scope of this work.

distribution of the simplified model indicates a translational component of the camera pose, which cannot be modeled by a single spherical coordinate system.

The compound errors of the individual models are mainly influenced by sample points of the camera frame. This indicates that the camera model used in this work is too simplistic. To lower the modeling accuracy and precision, a more general camera model can be applied.

The results presented in Sec. 4.1 show that a spherical representation of the RTS is sufficient for idealized geometric simulation of the system. If a more detailed model is required, the system can be described by DH parameters using the method we introduced in Sec. 3.1. The results proposed in Sec. 4.6 show the significance of the downstream numerical optimization.

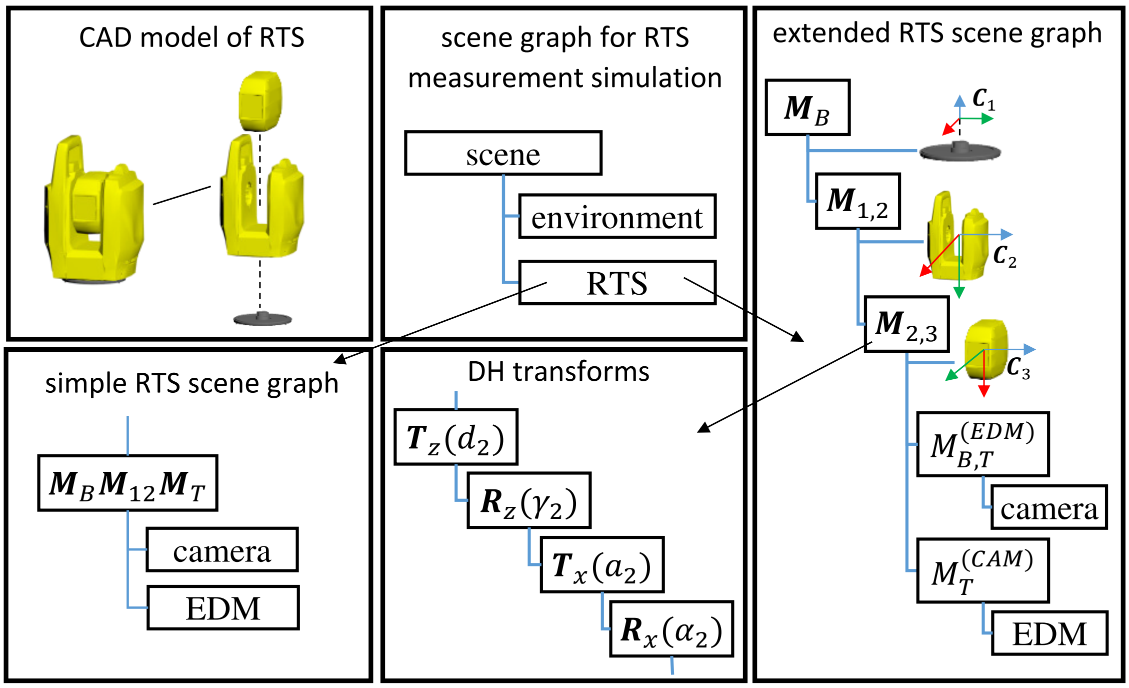


Figure 4.5: Disassembling an RTS to identify geometric frames, sensors and actuators for simulation.

The estimated models can be used for RTS simulation, using custom or standard robotic simulators. Figure 4.5 shows the scene graph of an exemplary RTS simulator, Figure 4.6 shows a custom RTS simulator in Unity3D.

The following section provides a detailed analysis of the simulator shown in Fig. 4.6. In particular, we use the simplified device model shown in Fig. 4.1, and the corresponding simple RTS scene graph as shown in Fig. 4.5 for the following discussions.

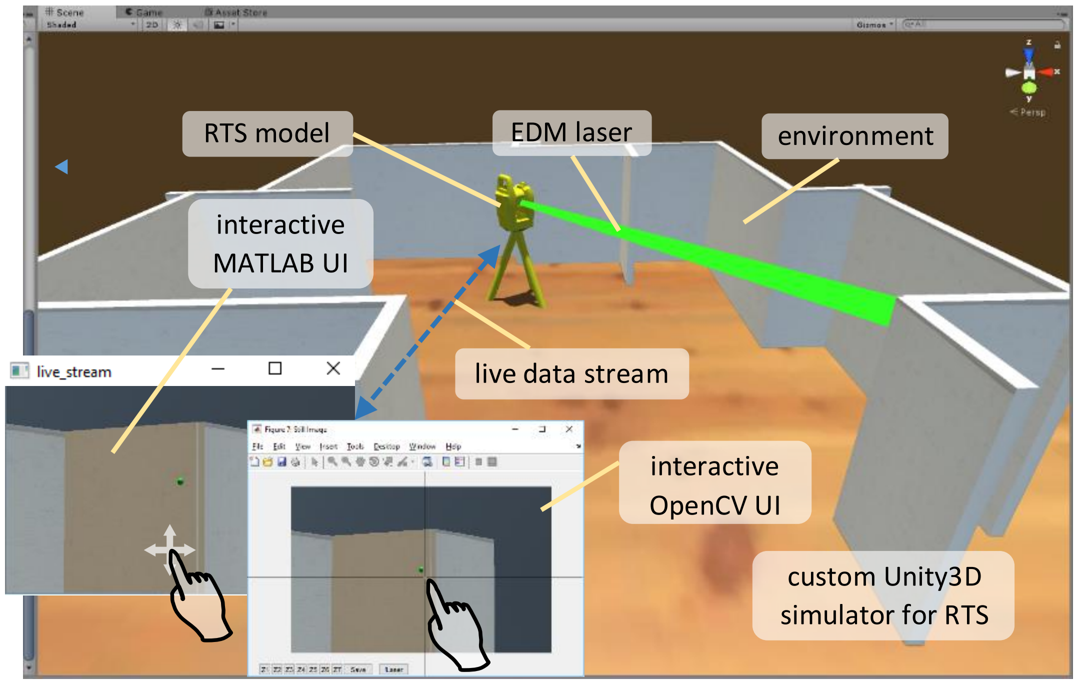


Figure 4.6: Exemplary RTS simulator, with Unity3D as simulation environment, the gRPC library [41] as communication and control layer, and two simultaneously connected clients for testing workflows and algorithms. While Unity3D uses mainly C#, the clients implement workflow and UI prototypes in C++ and MATLAB, respectively.

## 4.8 Using Unity3D as RTS Simulation Engine

3D rendering engines can be used for real-time simulation of various measurement setups. In this work, we use the game engine Unity3D [105] for creating and simulating measurement setups with scene graphs.

A Unity3D scene graph for RTS setups consists of nodes for sensors, actuators and measurement targets with attached *behavior* scripts [105]. The simulation includes firmware behavior, environmental influences and timing constraints. Figure 4.6 shows an exemplary scene, and Figure 4.5 shows the corresponding scene graph.

### 4.8.1 Modeling RT Sensors, Actuators and Targets in Unity3D

The geometric environment of the scene graph shown in Figure 4.5 consists of multiple triangle meshes. The *instrument node* represents the reference frame for RTS measurements, which can be freely positioned within the scene. Telescope frame, camera sensor and EDM are modeled as child nodes of the instrument node. The *environment node* is a placeholder for different measurement targets. Environments may include CAD models of reflective and non-reflective measurement targets, individual rooms, complete buildings or urban areas. The EDM is modeled by ray casting and returns the smallest distance $d$

between the ray origin and the closest ray intersections with a scene object. The reader is referred to the Unity3D User Manual [105] for implementation details.

### 4.8.2 Converting Coordinate System Handedness

Unity3D uses a left-handed coordinate system [105], while most RTS models use right-handed coordinate systems [9, 106]. Accordingly, the kinematic models derived in the previous section are based on right-handed coordinates systems to maintain consistency and better comparability with previous published work in this field. Hence, geometric models and simulation results of the Unity3D module must be converted accordingly. Implementation details are given in Appendix B.4.

### 4.8.3 Modeling Sensor Uncertainties in Unity3D

RTS manufacturers specify sensor uncertainties for normal distributed random variables in following general form:

$$p(|x' - x| \leq k u_c(x')) = CI_k \tag{4.12}$$

Here $x$, is the measured quantity, $u_c(x')$ is the combined standard uncertainty of the measurement result $x'$, $k$ is the coverage factor, and $CI_k$ is the corresponding confidence interval.

For RTS sensors, one can estimate the combined standard uncertainty according to:

$$u_c(x') \approx \sqrt{u_1(x')^2 + (x u_2(x'))^2} \tag{4.13}$$

Here, $u_1(x')$ is the bias and $u_2(x')$ is the scale, both provided by the device manufacturers or through sensor calibration. Usually, the true value $x$ is not known, and the current measurement $x'$ is used instead in Eqn. 4.13 to estimate the combined standard uncertainty $u_c(x')$. In this work, the error model given in Eqn. 4.12 is applied to simulate EDM and angular sensor uncertainties. The measurement uncertainty distribution is assumed to have normal distribution and zero mean. In addition, the servo actuator and angular sensor are split, thereby increasing the flexibility of simulation. Figure 4.7 shows the sensor uncertainty model of the RTS simulator, Figure 4.8 shows the verification setup for the simulation uncertainty. We use the Box-Muller transform [18] to generate the sensor reading $x'$ from uniformly distributed random values $a, b$ and the simulated true value $x$:

$$x' = x + \sqrt{-2ln(a)} \cos\left(2\pi b u_c(x')\right) \tag{4.14}$$

Here, the desired standard uncertainty $u_c(x')$ is taken from Eqn. 4.13.

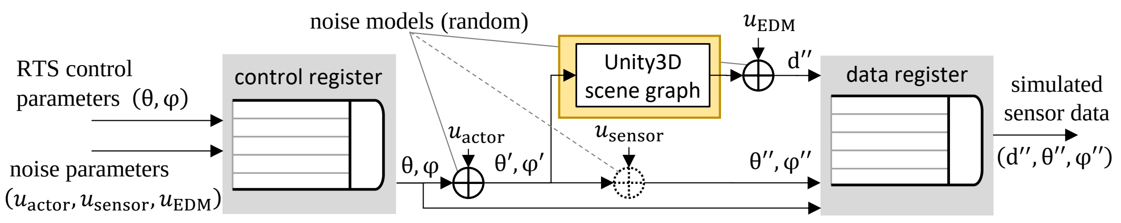


Figure 4.7: Simulating sensor uncertainties for RTS, where the $u_{actuator}$ and $u_{sensor}$ are the servo and angle uncertainties, and $u_{EDM}$ is the distance uncertainty of the EDM. Different configurations of hardware and environmental conditions can be simulated by altering the noise parameters.

## 4.9 Uncertainties in Simulations

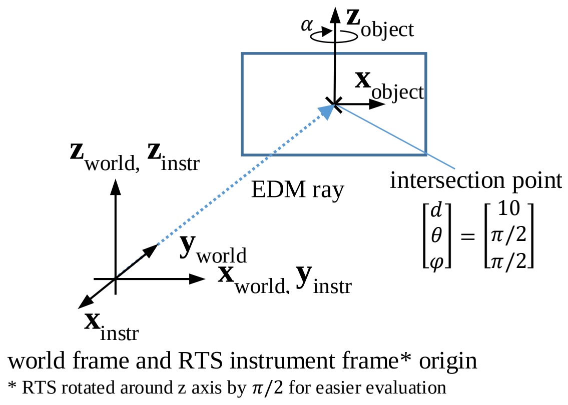
The analysis and report of measurement uncertainties of physical sensors is crucial for assessing simulation and measurement results. In Sec. 4.8.3, a simple model for sensor uncertainty simulation was provided, used for simulating realistic hardware sensors. However, the limitations of the real-time simulation itself have not been considered yet.

The JCGM 100:2008 GUM [57] standardizes the evaluation and report of measured physical quantities, using measurement uncertainties to guarantee reliable and repeatable experiments. In this section, we analyze the simulation uncertainties of virtual RTS experiments as shown in Fig. 4.9. The applied methods for estimating the simulation uncertainty are conform to GUM.

We will derive a simple a-priori estimate of the uncertainty of a particular simulation setup. For this particular investigation, we assumed the ideal geometric model presented in Sec. 4.3 and in Sec. 4.4, without any systematic modeling error and with ideal sensors. Hence, all sensor uncertainties discussed in Sec. 4.8.3 were set to zero. As a result, only the variability of the simulation in Unity3D itself is considered here. Unless otherwise stated, arithmetic rules, naming convention and format specification conform to the IEEE 754-2008 standard for floating-point arithmetic [51] and follow GUM [57].

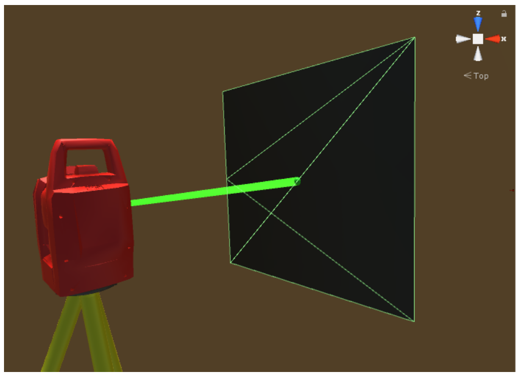
### 4.9.1 Identifying Sources of Uncertainty in Simulations

Unity3D allows the placement of scene objects anywhere in the coordinate system. Unlike CAD tools, game engines rely on fast single-precision (32-bit) floating-point representations for handling geometric entities. In this work, we use 64-bit floating-point format when working with the real device and as a data interchange format, but the 32-bit floating-point format of Unity3D for scene graph simulation and rendering. Hence, we must consider rounding errors of numerical operations on mesh vertices and scene objects with large distances to the world frame origin.

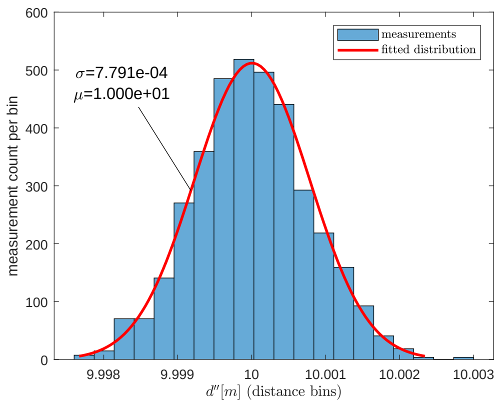


$\alpha$
$\mathbf{z}_{\text{object}}$
$\mathbf{x}_{\text{object}}$
$\mathbf{z}_{\text{world}}$, $\mathbf{z}_{\text{instr}}$
EDM ray
intersection point
$\begin{bmatrix} d \\ \theta \\ \varphi \end{bmatrix} = \begin{bmatrix} 10 \\ \pi/2 \\ \pi/2 \end{bmatrix}$
$\mathbf{y}_{\text{world}}$
$\mathbf{x}_{\text{world}}$, $\mathbf{y}_{\text{instr}}$
$\mathbf{x}_{\text{instr}}$
world frame and RTS instrument frame* origin
* RTS rotated around z axis by $\pi/2$ for easier evaluation

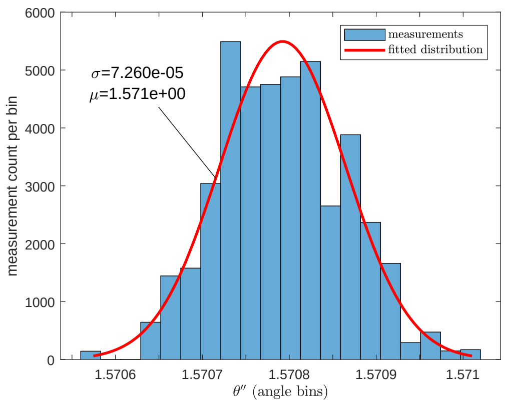
(a)



(b)



(c)



(d)

Figure 4.8: Unit test setup for verifying the simulated sensor noise. (a) Concept for uncertainty verification. (b) Unity3D setup for verifying both, simulated sensor uncertainty (desired) and round-off effects of the simulation (undesired). (c) Exemplary values for distance measurement uncertainties: $u_1(x\prime) = 0.75\times10^{-3}$, $u_2(x\prime) = 10\times10^{-6}$. (d) Exemplary values for angle sensor uncertainties: $u_1(x\prime) = 5'' \approx 2.4241\times10^{-05} rad$, $u_2(x\prime) = 0$.

### 4.9.2 A-Priori Uncertainty Estimation

Figure 4.9 shows an example simulation setup with metric scale and realistic object positions. The simulator follows a graphics pipeline [95] as shown in Fig. 4.10. In this work, we focus on geometric operations of triangles only, but do not analyze the simulation uncertainty for rendering 2D images[5].

The uncertainty of geometric operations on triangles are affected by three major blocks (Figure 4.11): *Block I* transforms CAD vertices and normals $(\mathbf{x}, \mathbf{n})$ to the word frame;

[5]The image resolution is considered as the main pixel uncertainty source for both, the physical device and the simulation. An extended analysis of the image uncertainty is provided in the work by Erhart [31].

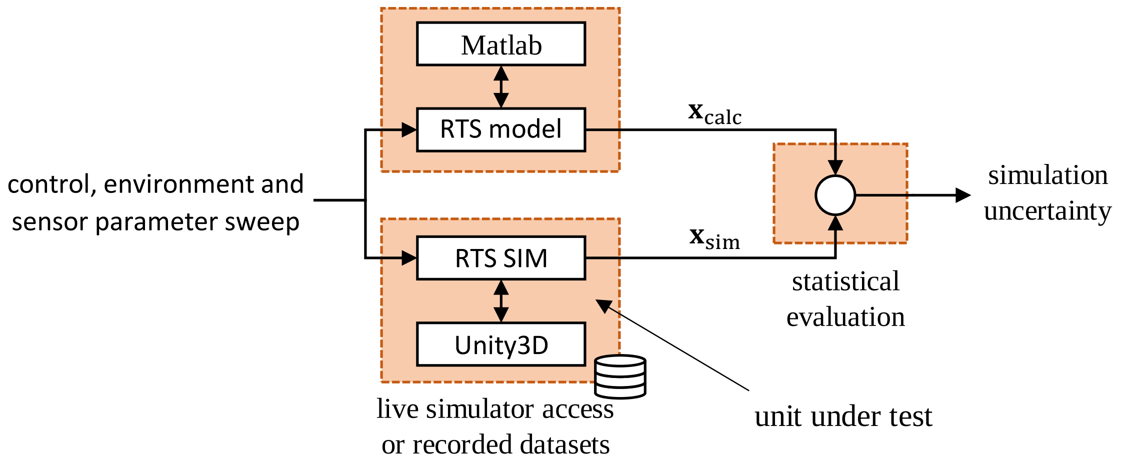

Matlab
RTS model
control, environment and sensor parameter sweep
RTS SIM
Unity3D
$\mathbf{x}_{\text{calc}}$
$\mathbf{x}_{\text{sim}}$
simulation uncertainty
statistical evaluation
live simulator access or recorded datasets
unit under test

Figure 4.9: Simple scene setup for the uncertainty analysis of floating-point effects.

*Block II* transforms the RTS object to the world frame; *Block III* calculates the EDM distance using ray casting. The ray casting result was interpreted as spherical coordinate vector with optional conversion to the Euclidean space. The latter was carried out with 64-bit floating-point arithmetic, which we regarded as having negligible error for our purposes. For better readability, the enumeration indices of vertices and normals were omitted in the following.

Input vertices are originally stored at 64-bit precision, but loaded in *block I* with 32-bit

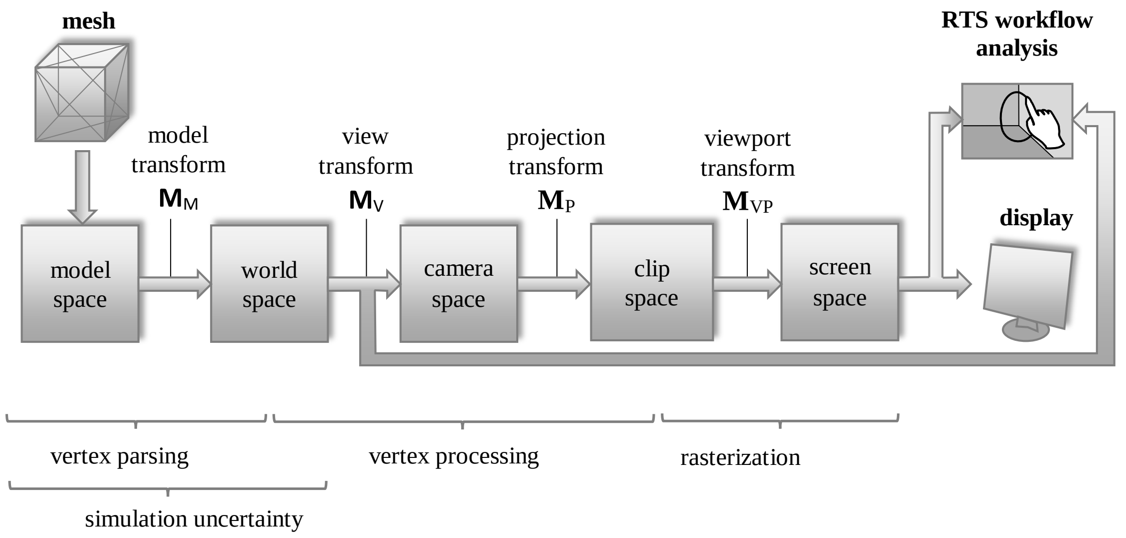


Figure 4.10: Graphics pipeline for triangles [95]. In this work, geometric operations on triangles that are relevant for the simulation uncertainty analysis are carried out in world-space. Therefore, only the first transformation of the vertex transformation pipeline needs to be considered.

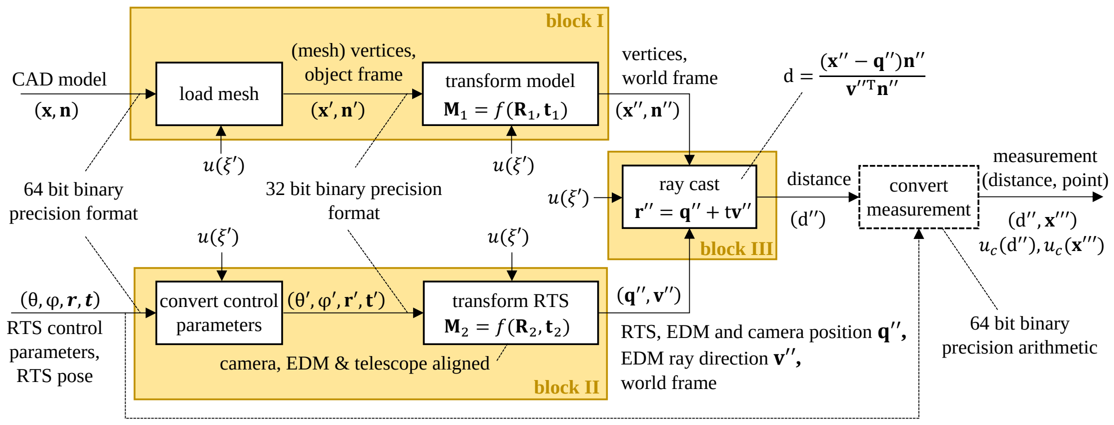

block I
CAD model
$(\mathbf{x}, \mathbf{n})$
load mesh
(mesh) vertices, object frame
$(\mathbf{x}', \mathbf{n}')$
transform model
$\mathbf{M}_1 = f(\mathbf{R}_1, \mathbf{t}_1)$
vertices, world frame
$(\mathbf{x}'', \mathbf{n}'')$
$d = \frac{(\mathbf{x}'' - \mathbf{q}'')\mathbf{n}''}{\mathbf{v}''^T\mathbf{n}''}$
$u(\xi')$
64 bit binary precision format
32 bit binary precision format
ray cast
$\mathbf{r}'' = \mathbf{q}'' + t\mathbf{v}''$
block III
distance
$(d'')$
convert measurement
measurement (distance, point)
$(d'', \mathbf{x}''')$
$u_c(d''), u_c(\mathbf{x}''')$
$(\theta, \varphi, \boldsymbol{r}, \boldsymbol{t})$
RTS control parameters, RTS pose
convert control parameters
$(\theta', \varphi', \mathbf{r}', \mathbf{t}')$
transform RTS
$\mathbf{M}_2 = f(\mathbf{R}_2, \mathbf{t}_2)$
$(\mathbf{q}'', \mathbf{v}'')$
camera, EDM & telescope aligned
block II
RTS, EDM and camera position $\mathbf{q}''$, EDM ray direction $\mathbf{v}''$, world frame
64 bit binary precision arithmetic

Figure 4.11: Estimating the round-off effects for simulations with uncertainty propagation. The critical aspect in this work is the EDM simulation, here modeled as ray casting pipeline with three connected processing blocks.

floating-point precision. The expected rounding error is mainly influenced by the distance between vertex and origin and the limits of the data format. Figure 4.12 shows the memory layout of the 32-bit binary floating-point format, including 1-bit sign, an 8-bit exponent, and a 23-bit *mantissa* plus one implicit leading bit. If the *format precision p* denotes the maximum number of digits at radix (base) 10, which can be represented, vertices and normals in Unity3D have a format precision of $p = 7$.

Let $\xi$ be an input number at radix 10. The rounded floating-point format $\xi'$ and the round-off error $e_\xi$ are given as follows:

$$\xi' = \lfloor \xi \cdot 10^{N_{frac}} \rceil \cdot 10^{-N_{frac}} \qquad e_\xi = \xi - \xi' \tag{4.15}$$

$$N_{frac} = p - N_{int} \qquad N_{int} = \begin{cases} \lfloor \log_{10} |\xi| \rfloor + 1, & \xi \neq 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.16}$$

Here, $N_{int}$ and $N_{frac}$ are the number of integer and fractional digits, respectively; $\lfloor x \rceil$ and $\lfloor x \rfloor$ round a real value $x$ to the nearest integer and towards minus infinity, respectively. If the true value $\xi$ is not known, the uncertainty bounds $a_+ - a_- = 2a$ of the rounded vertex $\xi'$ are given by a rectangular distribution:

$$\xi - a_- \leq \xi' \leq \xi + a_+ \qquad a = 0.5 \cdot 10^{-(N_{frac}+1)} \tag{4.17}$$

Here, $a_-$ and $a_+$ are the lower and upper limit, respectively. The standard uncertainty of a simulated vertex without any applied transformation is then given as follows [57]:

$$u(\xi') \approx \frac{0.5}{\sqrt{3}} \cdot 10^{-(N_{frac}+1)} \tag{4.18}$$

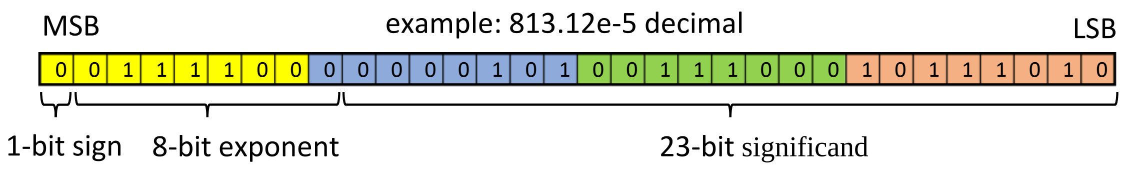


Figure 4.12: Memory layout and bit allocation of 32-bit floating-point numbers according to IEEE 754-2008 standard for floating-point arithmetic [51].

Let $\mathbf{x}$ be an input vertex $\mathbf{x}$ with corresponding normal $\mathbf{n}$, given in 64-bit. Let $\mathbf{x}'$ and $\mathbf{n}'$ be the corresponding entities in 32-bit floating-point precision. Then, the uncertainties for each element of the vertex and normal are given as follows:

$$u(x') \approx u(\xi')|_{\xi=x_b} \qquad u(n') \approx u(\xi')|_{N_{frac}=7} \tag{4.19}$$

Here, $u(x')$ defines the element-wise uncertainties of a vertex $\mathbf{x}'$, and $u(n')$ defines the element-wise uncertainties of the normal $\mathbf{n}'$. A conservative approximation for all vertices is given by $\xi = x_b$, where $x_b$ is the maximum absolute value of all components of the scene-bounding box. Assuming $||\mathbf{n}|| = 1$, the fractional digit count for the normals in Eqn. 4.19 is given by $N_{frac} = 7$.

To reduce the calculation complexity, we used following approximations: Scene transformations were reduced to Euclidean transformations to avoid homogeneous matrix operations. Where feasible, concatenated transformations as a single transformation only.

The quadratic variance propagation for a single output and multiple input variables was used to combine the uncertainty sources, which can be written in the following form [11, 102]:

$$u_c(\chi) = \sqrt{\mathbf{g}^T \mathbf{V} \mathbf{g}} \qquad \mathbf{g} = \nabla\chi = \left[\frac{\delta\chi}{\beta_1} \dots \frac{\delta\chi}{\beta_N}\right] \qquad \boldsymbol{\beta} = (\beta_1, \dots, \beta_N) \tag{4.20}$$

Here, $u_c(\chi)$ is the combined standard uncertainty of some function $\chi(\boldsymbol{\beta})$, and $\mathbf{V}$ is the covariance matrix for $N$ input variables $\beta_i$ with $i \in \{1 \dots N\}$.

The transformation of a model space vertex $\mathbf{x}'$ to the corresponding world space vertex $\mathbf{x}''$ is given as follows:

$$\mathbf{x}'' = \mathbf{R}'\mathbf{x}' + \mathbf{t}' \qquad \mathbf{n}'' = \mathbf{R}\mathbf{n}' \tag{4.21}$$

Here, $\mathbf{R}'$ is the 32-bit floating-point representation of a $3 \times 3$ rotation matrix $\mathbf{R}$; $\mathbf{t}'$ is the 32-bit floating-point representation of a $3 \times 1$ translation vector $\mathbf{t}$, and $(\mathbf{x}', \mathbf{n}')$ are the 32-bit floating-point representations of the vertex and normal $(\mathbf{x}, \mathbf{n})$. To reduce the uncertainty transfer function output for block I to a single variable, the same scalar uncertainty $u_c(x'')$ was used for the $x$, $y$ and $z$ component of vertex $\mathbf{x}''$, and all components were treated as independent. Analogously, the translational standard uncertainty $u(t) \approx u(x')$ was assumed to be equal for all axes. The fractional digit count $N_{frac}$ in Eqn. 4.18 can be estimated using $|x| = x_b$.

Let $r'_{ij}$ be the elements of the rotation matrix $\mathbf{R}'$, $u(r_{ij})$ be the element-wise standard uncertainty, and $r_{ij}$ the elements of the true rotation matrix $\mathbf{R}$. Rotation matrix rows are normalized to one, which leads to $|r_{ij}| \leq 1$. Consequently, the uncertainty value for each matrix element $r'_{ij}$ can be approximated by $u(r'_{ij}) \approx u(\xi)$ with $N_{frac} = 7$, as defined in Eqn. 4.18. Without known rotation parameters, a liberal approximation for the variance propagation of the rotational part is given by substituting the identity matrix as rotation matrix $\mathbf{R}$. However, a conservative approximation is preferable, substituting $|r_{ij}| = 1$ for all matrix elements[6]. Consequentially, the standard uncertainty for each element of $\mathbf{x}''$ can be estimated from the first component of $\mathbf{x}''$ only, which is given by the first row of Eqn. 4.21 according to

$$x''_1 = r'_{11}x'_1 + r'_{12}x'_2 + r'_{13}x'_3 + t'_1 \qquad \mathbf{x}' = \begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix} \qquad \mathbf{t}' = \begin{bmatrix} t'_1 \\ t'_2 \\ t'_3 \end{bmatrix} \tag{4.22}$$

where $x''_1$ is the first component of vertex $\mathbf{x}''$.

According to Eqn. 4.20, the partial derivatives of $x''_1$ with respect to $\boldsymbol{\beta}$ can be calculated as follows:

$$\mathbf{g} = \nabla x''_1 \qquad \boldsymbol{\beta} = \left(x'_1 \ldots x'_3, r'_{11} \ldots r'_{13}, t'_1\right) \tag{4.23}$$

Here, $\boldsymbol{\beta}$ is the tuple of 7 scalar variables. The corresponding $[7 \times 7]$ covariance matrix $\mathbf{V}$ is given as follows:

$$\mathbf{V} = diag(\mathbf{1}_3^T u(x'), \mathbf{1}_3^T u(r'_{ij}), \mathbf{1}_3^T u(t')) \tag{4.24}$$

Here, $\mathbf{1}_3^T$ is a $1 \times 3$ vector with all elements equal one; all input variables are defined as independent; hence, all covariance elements are zero. By substituting $|x_i| = x_b$, $|r_{ij}| = 1$, the estimation of $u_c(x'')$ can be reduced to the following:

$$u_c(x'') \approx \frac{1}{k}\sqrt{3u(r'_{ij})^2 x_b^2 + 3u(x')^2 + u(t')^2} \tag{4.25}$$

Here, the regularization term $k$ takes the conservative assumptions into account; hence, it can be interpreted as *coverage factor*. Using $k = 3$ avoids overestimation of uncertainty effects and turns the expanded uncertainty of Eqn. 4.25 into the standard uncertainty as defined in GUM [57].

The element-wise combined standard uncertainty $u_c(n'')$ of a transformed vertex nor-

[6]This leads to an invalid rotation matrix, but the results are more trustworthy for conservative approximations without explicit knowledge of the rotation parameters.

mal $\mathbf{n}''$ can be directly derived from Eqn. 4.25. From $||\mathbf{n}'|| \approx 1$ follows that the bounding box for all normal elements in Eqn. 4.25 can be set to $|x_b| = 1$. Normals are not affected by translation; hence, $u(t')$ in Eqn. 4.25 is zero, and $u_c(n'')$ can be estimated as follows:

$$u_c(n'') \approx \frac{1}{k}\sqrt{3u(r'_{ij})^2 + 3u(n')^2} \tag{4.26}$$

*Block II* models the EDM by transforming the RTS control parameters to a ray in world-space. Let the instrument space ray be defined by the ray origin $\mathbf{q}$ and the ray direction $\mathbf{v}$. We approximated the chain of RTS transformations by a single Euclidean transformation as done in *block I*[7]. Assuming the same bounding box for all vertices, scene objects and the RTS position, the combined uncertainty values for the transformed world-space ray origin $\mathbf{q}''$ and the transformed world-space ray direction $\mathbf{v}''$ can be approximated by the combined uncertainty values calculated for block I:

$$u_c(q'') \approx u_c(x'') \qquad u_c(v'') \approx u_c(n'') \tag{4.27}$$

Here, $u_c(q'')$ and $u_c(v'')$ are the element-wise combined standard uncertainties for the transformed ray origin and ray direction, respectively.

For the analysis of *Block III*, the EDM ray casting was simplified to plane-ray intersection. The simulated distance $d''$ between the ray origin $\mathbf{q}''$ and the scene intersection point can be calculated as follows:

$$d'' = \frac{(\mathbf{x}'' - \mathbf{q}'')^T\, \mathbf{n}''}{\mathbf{v}''^T \mathbf{n}''} = \frac{(\mathbf{x}'' - \mathbf{q}'')^T\, \mathbf{n}''}{||\mathbf{v}''||\ ||\mathbf{n}''|| \cos\alpha} \tag{4.28}$$

Here, $d''$ is the ray length, calculated with finite precision arithmetic, $\mathbf{n}''$ is the plane normal, $\mathbf{x}''$ is a point on the plane, and $\alpha$ is the incident angle between the ray direction and the plane normal. The propagated combined standard uncertainty $u_c(d'')$ can be estimated by solving Eqn. 4.20 with $\chi = d''$, using the input parameters $\boldsymbol{\beta}$:

$$\boldsymbol{\beta} = \left(\mathbf{x}'', \mathbf{n}'', \mathbf{q}'', \mathbf{v}''\right) \tag{4.29}$$

Here, $\boldsymbol{\beta}$ is interpreted as tuple of 12 scalar variables. The corresponding $[12 \times 12]$ covariance matrix $\mathbf{V}$ is given as follows:

$$\mathbf{V} = diag(\mathbf{1}_3^T u_c(x''), \mathbf{1}_3^T u_c(n''), \mathbf{1}_3^T u_c(q''), \mathbf{1}_3^T u_c(v'')) \tag{4.30}$$

The denominator of Eqn. 4.28 shows a significant dependency between the combined uncertainty of the distance measurement, $u_c(d'')$, and the incident ray angle $\alpha$. A first or second order Taylor approximation of the denominator could be used for angles close to zero, but would be insufficient for larger angles. As an alternative, a rough estimation

[7]A more detailed, but also more complex approximation of the variance propagation would consider all control parameters in block II as shown in Figure 4.11.

of $u_c(d'')$ can be derived by substituting an independent variable $g$ with zero variance as denominator. Hence, Eqn. 4.28 can be written in the following form:

$$d'' = \frac{(\mathbf{x}'' - \mathbf{q}'')^T \mathbf{n}''}{g} \qquad ||\mathbf{v}''|| \approx ||\mathbf{n}''|| \approx 1 \Rightarrow g \approx \cos(\alpha) \tag{4.31}$$

Using the same considerations as for blocks I and II, $u_c(d'')$ can be approximated by solving Eqn. 4.20 with $\chi = d''$, where $\boldsymbol{\beta}$, $\mathbf{V}$ and $d''$ are given in Eqns. 4.29 - 4.31, respectively. The conservative substitutions of $|n''_i| = 1$ for all elements of $\mathbf{n}$ and $\mathbf{x}'' - \mathbf{q}'' = \begin{bmatrix} 2x_b & 2x_b & 2x_b \end{bmatrix}^T$ lead to the following form:

$$u_c(d'') \approx \frac{1}{|\cos(\alpha)|} \sqrt{12 u_c(n'')^2 x_b^2 + 3 u_c(x'')^2 + 3 u_c(q'')^2} \tag{4.32}$$

Here, $x_b$ denotes the maximum absolute element of the scene-bounding box.

Optionally, the simulated distance measurement $d''$ and the angle control parameters $(\theta, \varphi)$ can be converted to an Euclidean point $\mathbf{x}'''$ using Eqn. 4.4 with 64-bit floating-point arithmetic. If the norm of all columns of an Euclidean rotation matrix equals one, the combined uncertainty for each element of the measured point can be estimated as follows:

$$u_c(x''') \approx m \frac{u_c(d'')}{\sqrt{3}} \qquad 1 \leq m \leq \sqrt{3} \tag{4.33}$$

Here, $m$ is a correction factor that avoids underestimation for special scene setups. For example, when measuring far distances along a single axis, the combined standard uncertainty $u_c(x''')$ of the significant axis can be analyzed with $m = \sqrt{3}$.

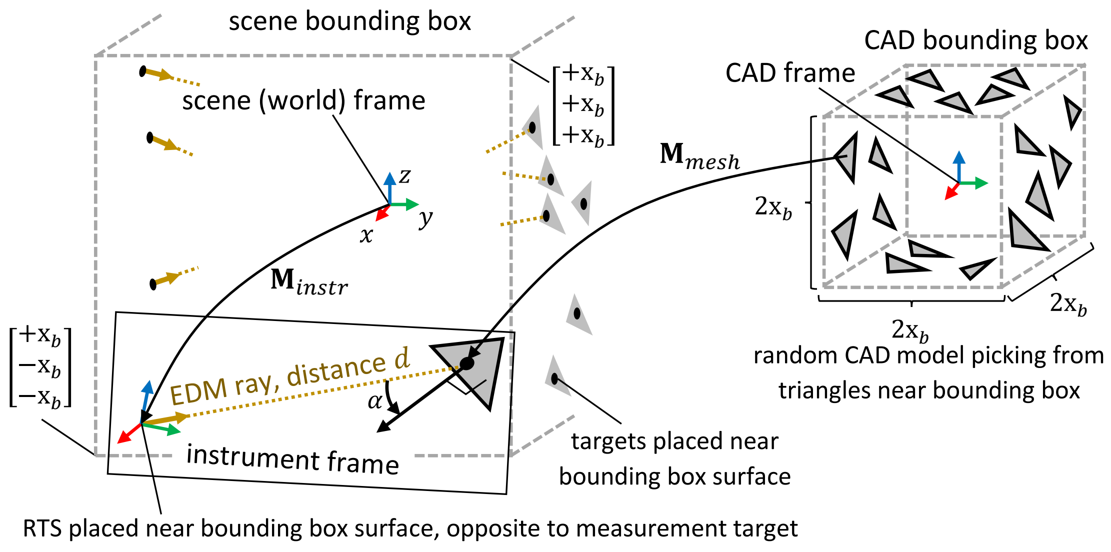
## 4.10 A-Posteriori Uncertainty Estimation

We evaluated our setup with Monte-Carlo Simulation (MCS) based on random placements of the simulated RTS and associated targets.

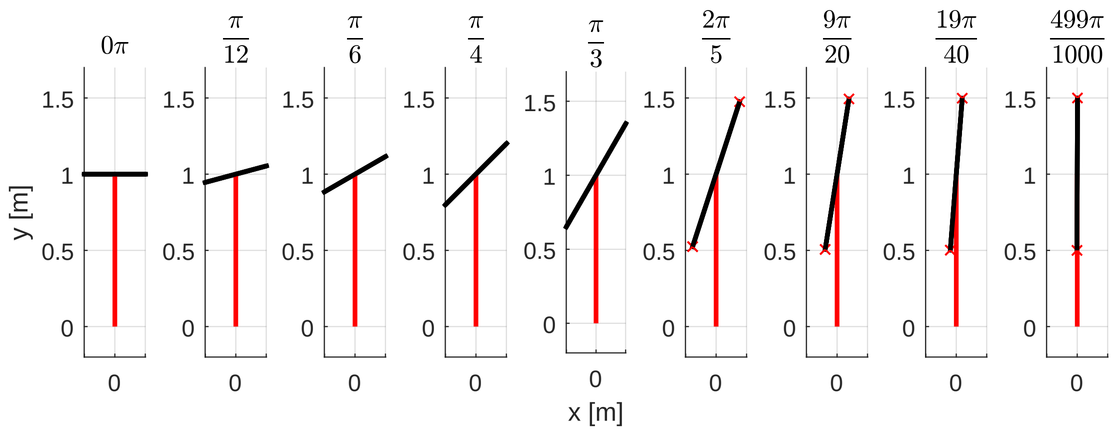
The uncertainty propagation functions developed in Sec. 4.9.2 can be verified using MCS. Let $\mathbf{y}_i$ be the true intersection point of a laser distance measurement, and, $\mathbf{q}_i$, the laser ray origin, both calculated with 64-bit arithmetic. The standard uncertainty $u(d'')$ for the simulated distance measurement can be estimated by $N$ repeated measurements, using the following:

$$u(d'') \approx \sqrt{\frac{(d''_i - d_i)^2}{N}} \qquad d_i = ||\mathbf{y}_i - \mathbf{q}_i|| \tag{4.34}$$

The simulator setup for the MCS is shown in Figure 4.13. For each experiment, the CAD model, RTS pose and measurement target pose were randomly generated. The CAD model of each experiment consists of a single triangle, the measurement target, with arbitrary


scene bounding box
scene (world) frame
$\begin{bmatrix}+x_b\\+x_b\\+x_b\end{bmatrix}$
$\begin{bmatrix}+x_b\\-x_b\\-x_b\end{bmatrix}$
$z$
$y$
$x$
$\mathbf{M}_{instr}$
$\mathbf{M}_{mesh}$
EDM ray, distance $d$
$\alpha$
instrument frame
CAD bounding box
CAD frame
$2x_b$
$2x_b$
$2x_b$
random CAD model picking from triangles near bounding box
targets placed near bounding box surface
RTS placed near bounding box surface, opposite to measurement target

(a)



(b)

Figure 4.13: Setup for analyzing the uncertainty of arithmetic floating-point effects using MCS. (a) Relations, transformations and parameters of scene objects, object frames, and ray intersections. (b) Experiments with different incident angles of the EDM ray at the measurement target, here shown in 2D for simplicity reasons.

rotation, a circumradius of one, and the centroid placed randomly on the bounding box. In the world-space frame, the RTS was placed on the negative and the measurement target was placed on the positive $y = 0$ plane of the bounding box. Each experiment was evaluated using the simulator and compared with results of a 64-bit precision arithmetic model.

Table 4.7: Uncertainty results for the realistic RTS simulator setup, using analytical method and MCS. For the analytical estimation, a scene-bounding box of $x_b \approx 10$m was used. Intermediate variables of the uncertainties are not available because the simulator API was directly generated from the original RTS driver API and the physical setup. Therefore, $u(n')$, $u(x')$ and $u(n'')$ where not accessible (NA) in this experiment.

| $\alpha$ | method | $u(n')$ $[m]$ | $u(x')$ $[m]$ | $u_c(n'')$ $[m]$ | $u_c(x'')$ $[m]$ | $u_c(d'')$ $[m]$ |
|---|---|---|---|---|---|---|
| $0.33\pi$ | analytical | $2.89\times10^{-8}$ | $5.00\times10^{-7}$ | $2.36\times10^{-8}$ | $3.44\times10^{-7}$ | $1.87\times10^{-6}$ |
| | MCS | NA | NA | NA | NA | $2.41\times10^{-6}$ |
| $0.45\pi$ | analytical | $2.89\times10^{-8}$ | $5.00\times10^{-7}$ | $2.36\times10^{-8}$ | $3.44\times10^{-7}$ | $6.00\times10^{-6}$ |
| | MCS | NA | NA | NA | NA | $7.34\times10^{-6}$ |
| $\approx 0.46\pi$ | MCS realistic | NA | NA | NA | NA | $5.82\times10^{-6}$ |

### 4.10.1 Interpretation of Uncertainty Estimation Results

We estimated the element-wise combined standard uncertainties $u_c(x''')$ of the intersection point $x'''$ for the distances $d \in \{0.5, 1, 10^1, 10^2, 10^3\}$ meter. The world-space bounding box was assumed to be $x_b \approx \pm\frac{d}{2}$ in $x$, $y$ and $z$ direction, respectively. The RTS was placed near the border of the bounding box as shown in Figure 4.13. The rays' incident angles were set to $\alpha \in \{0, \frac{\pi}{12}, \frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3}, \frac{2\pi}{5}, \frac{9\pi}{20}, \frac{9\pi}{40}, \frac{9\pi}{1000}\}$. The Euclidean transformations $\mathbf{M}_{instr}$ and $\mathbf{M}_{mesh}$ describe the world-space transformation for the RTS and the measurement target, respectively. For each experiment, both matrices were randomly generated, placing the RTS and the measurement target near the scene-bounding box. The CAD model for each experiment consisted of single triangle, the measurement target, randomly placed within the CAD bounding box. Each triangle was created by randomly picking three points near the CAD bounding box. In addition, $\mathbf{M}_{instr}$ and $\mathbf{M}_{mesh}$ include jitter for the poses of each experiment; the jitter was created using random translations with $\pm0.5$m in $x$,$y$ and $z$ direction and random rotations with $\pm0.05$rad for each Euler rotation angle. In addition, a local RTS rotation around the $z$-axis of the instrument frame was applied, using randomized angles between 0 and $2\pi$. Figure 4.14 shows the resulting uncertainty characterization curves, estimated analytically and by using MCS. The run-time of the analytical uncertainty evaluation was $4.2\times10^{-3}s$, implemented and executed with MATLAB 2017, under Microsoft Windows 10 on an Intel Core i7 processor with 64GB RAM. The MCS took $2.5\times10^{3}s$ ($\approx 0.7h$), using 100 samples for each configuration.

In addition, we estimated the simulation uncertainty for the realistic measurement setup shown in Figure 4.9. We assumed a scene-bounding box with $x_b \leq 10$m and expected incident ray angles of $\alpha \approx (\frac{\pi}{3}, 0.45\pi)$ in Eqn. 4.32. The uncertainty results of the MCS with the realistic setup showed an average incident angle of $0.46\pi$ and an average EDM distance of $7.2m$. Results are provided in Table 4.7.

The classification curves provided in Figure 4.14 allow uncertainty estimations prior to the design of simulation setups and related CAD models. The analysis shows a linear relationship between the logarithm of the simulation uncertainty and the logarithm of the scene-bounding box; furthermore, a significant influence of the indecent angle is observable. Figure 4.14e shows the overlay of the analytic approach and the MCS; Figure 4.14f shows the difference between the two methods. Both diagrams indicate that the analytical uncertainty estimation is a reasonable prior approximation. With increasing incident angle, the difference between the two methods increases.

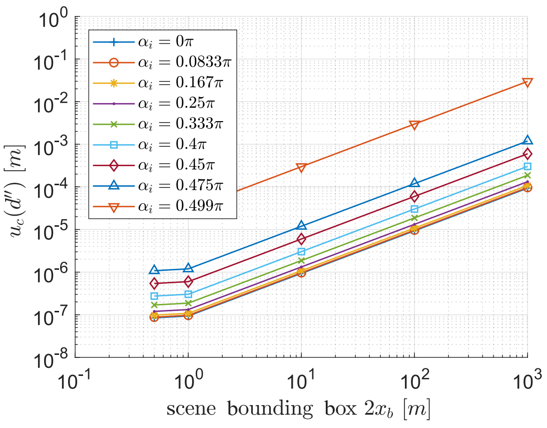
## 4.11 Discussion

In this chapter, we presented a Unity3D based RTS simulator for graphical and non-graphical algorithm development, test and verification, including a simple geometric models for the exemplary RTS, actuators, sensors and uncertainties. The provided analytical a-priori method can be used for assessing simulation uncertainties before the actual implementation of a particular measurement simulation.

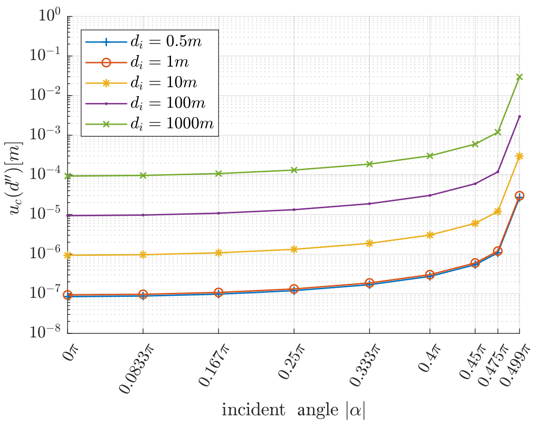
We validated the results that are proposed in this chapter with MCS, which provide more detailed uncertainty estimations for a particular setup, but at the cost of increased execution time.

Furthermore, we verified the results of the analytical a-priori method using a realistic simulation setup.

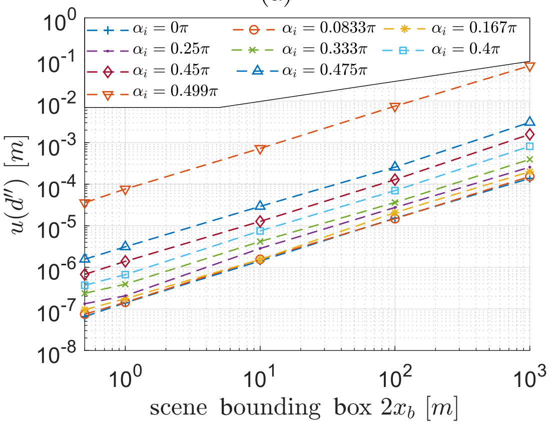
In the following chapters, we describe two assistance systems for reflectorless measurement tasks. The simulation concept proposed in this chapter was an essential part for the design, implementation and test of the proposed workflows.



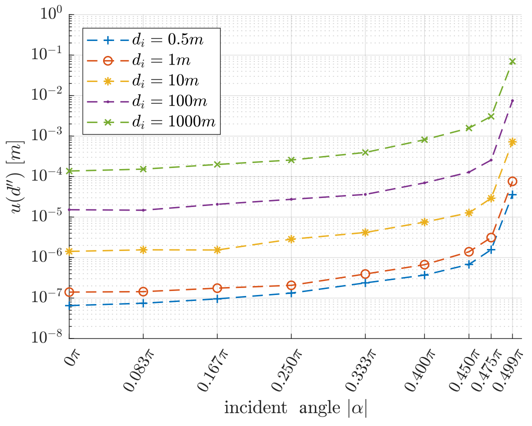
(a)



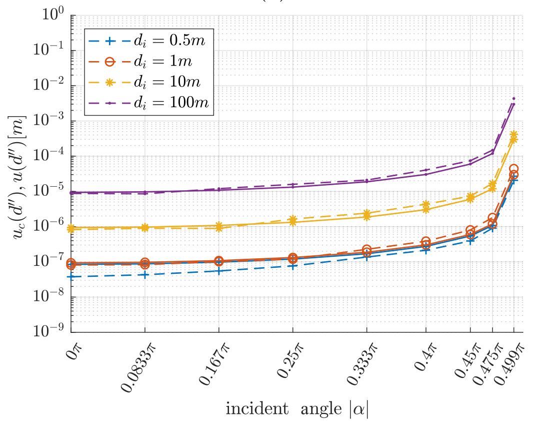
(b)



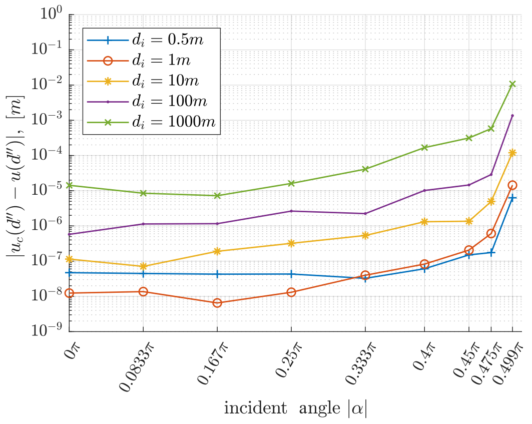
(c)



(d)



(e)



(f)

Figure 4.14: Simulation uncertainty estimations of EDM and target intersections in Unity3D. (a) Analytically estimated uncertainty with respect to the scene-bounding box. (b) Analytically estimated uncertainty with respect to the incident angle. (c) Uncertainty from MCS, plotted with respect to the measured scene-bounding box. (d) Uncertainty from MCS, plotted with respect the incident angle. (e) Analytical uncertainty $u_c(d'')$ (continuous lines) overlaid by MCS (dashed lines) for better comparison. (f) Difference between analytical uncertainty $u_c(d'')$ and MCS.

# 5 Application: Measuring with Support Objects

Modern RTS support reflectorless measurements using the diffuse reflection of natural surfaces. This is often referred to as measuring *natural targets*. Common natural targets in surveying and building construction should have a high recall value; thus, preferred targets are corners and edges of human-made structures. However, measuring non-planar targets with an RTS in reflectorless mode is a challenging and error-prone task, as any accurate 3D point measurement requires a fully reflected laser beam of the electronic distance meter and proper orientation of the pan-tilt unit. Influences of the laser beam divergence of the EDM, angular resolution of the theodolite, inaccurate targeting and optical limitations are reasons why direct measurements of non-planar targets are critical. Surveyors often use post-processing methods to increase the accuracy of such measurements. Figure 5.1 shows the systematic error introduced by the aforementioned constraint. An extensive discussion about measuring non-planar targets with RTS is provided by Juretzko [60].

In this chapter, we present three algorithms and UI for simple and efficient construction-side measurement corrections of the systematic error, using additional measurements close to the non-measurable target. Post-processing of single-point measurements is not required with our methods, and our experiments prove that using a 3D point, a 3D line or a 3D plane support can lower the systematic error by almost a order of magnitude. As a side effect, the proposed approach simplifies the overall measurement procedure such that even non-experts in the field can perform reliable and robust measurements. This is proven by the results of our pilot study.

This chapter is based the work by Klug et al. [65, 67], but provides a more complete description.

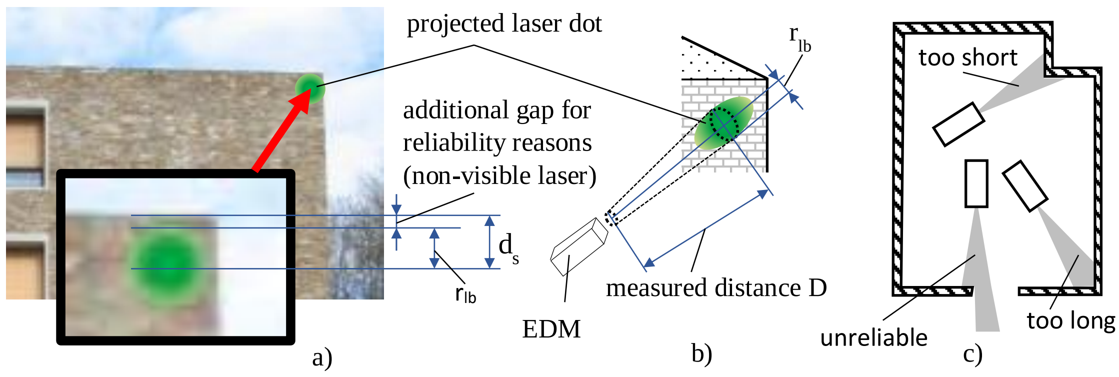


Figure 5.1: For reliable measurements of distances and 3D positions, the laser should be fully reflected by a planar surface of natural targets. In general, the measurement uncertainty increases when using non-planar targets. Similarly, the measurement reliability decreases when using partly-reflected laser beams as the EDM behavior is often not full specified (Juretzko [60]). When enforcing a fully-reflected laser with image-based targeting, the minimum vertical and horizontal measurement error $e$ can be approximated by $e = d_0 + r_{lb}$, where $r_{lb}$ is the radius of the projected laser beam, and $d_0$ is the *safety distance* between the edges of the target and the laser pointer. Note that $r_{lb}$ approximates the elliptical projection of the laser through by a circle. The safety distance $d_0$ is influenced by user experience, image resolution, focal length of the camera, image blur due to out-of-focus problems, back light conditions and other effects. An additional challenge is to correctly observe the projected laser dot on the surface, as the laser is often barely visible or not visible at all.

## 5.1 Extending the Measurement Workflow

When measuring a corner directly in reflectorless mode, the laser dot must be fully reflected by an attached surface. For corners and edges, this assumption is violated. Following countermeasures can be applied:

(a) Violate the assumption and risk distance measurement errors.
(b) Modify the corner physically to create at temporary planar surface.
(c) Measure multiple points close to the corner and select the best candidate manually.
(d) Measure multiple points close to the corner and apply geometric corrections.

The effects of violating the EDM specification and out-of-spec operation are not considered in this work. We temporary add a planar surface to the measurement targets to generate reference data for our tests. Such modifications can be used under laboratory conditions, but they are often impractical on construction sites for reasons of limited target accessibility, measurement effort and personnel costs. A common approach is not to measure the corner directly, but to measure a point close to the actual target. Hereby, the planar area around the point nearby is bigger than the projected laser area. However, this method adds and additional targeting error. Therefore, we define three alternative methods that integrate in-the-field corrections for corner and edge measurements to lower the targeting error. In this work, we compare following five methods:

(r) direct measurement method with physical changes of the target (direct method), used as reference in this work,
(a) nearby measurement method (nearby method),
(b) target measurement using a virtual support point (support point method),
(c) target measurement using a virtual support line (support line method), and
(d) target measurement using a virtual support plane (support plane method).

Figure 5.2 shows the direct and nearby measurement methods as well as the support point, support line and support plane method. Details about the methods are provided in following sections.

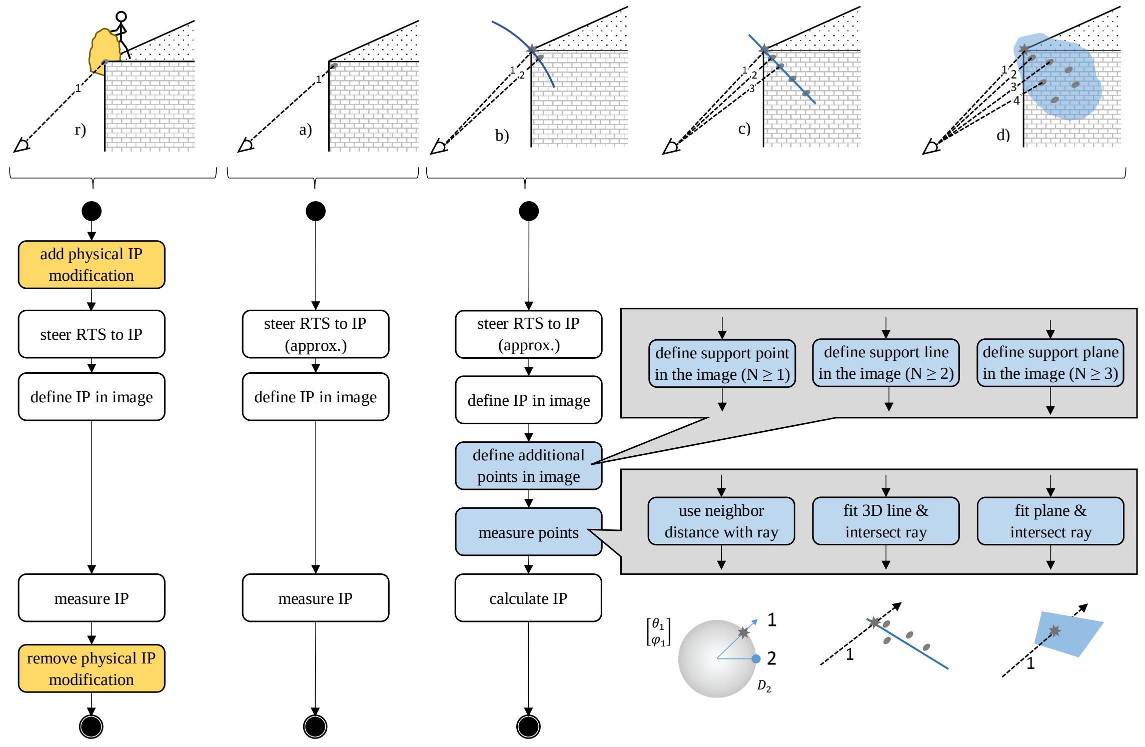

r)
a)
b)
c)
d)
1
2
3
4
add physical IP modification
steer RTS to IP
define IP in image
measure IP
remove physical IP modification
steer RTS to IP (approx.)
define IP in image
measure IP
steer RTS to IP (approx.)
define IP in image
define additional points in image
measure points
calculate IP
define support point in the image (N ≥ 1)
define support line in the image (N ≥ 2)
define support plane in the image (N ≥ 3)
use neighbor distance with ray
fit 3D line & intersect ray
fit plane & intersect ray
$\begin{bmatrix}\theta_1\\ \varphi_1\end{bmatrix}$
$D_2$

Figure 5.2: Five different measurement methods of a corner with a single visible adjacent area as proposed in [65, 67]: (a) direct method, (b) nearby method, (c) support point method, (d) support line method, and (e) support plane method. The view rays are enumerated according the measurement order used for our experiments.

## 5.1.1 Standard Methods: Direct and Nearby Method

In reflectorless mode, the EDM laser should fully hit the planar measurement target. Non-planar surfaces increase the measurement uncertainty; partly reflected laser beams additionally lower the measurement reliability.

From a mathematical point of view, the direct and nearby method are almost identical. When using the *direct method*, the user measures the IP directly, and the current angle and distance measurements are used for the conversion to a 3D point. Adding physical installations or using the *nearby method* are intuitive solutions if a fully reflected laser dot cannot be guaranteed. When using the *nearby method*, the user does not aim for the IP directly, but for a measurable point close to it. The benefit of using the nearby method is that the measurement uncertainty is user-controlled. Repeated measurements with slowly decreasing safety gap between the laser and the edge of the target allows an experienced user to decrease the measurement uncertainty.

#### 5.1.1.1 Measurement Flow

The simple measurement flow is defined by following steps:

1. Add physical target modification (optional)
2. Use the pan/tilt control interface, until the target-of-interest is visible in the image.
3. Define target-of-interest in the image.
4. Calculate the 3D position of the target-of-interest by measuring the angle and distance at the selected image point.
5. Remove physical target modification (optional)

#### 5.1.1.2 Calculating the Point

The simplified model of an RTS defines a local spherical coordinate system as described in Sec. 4.3 and Sec. 4.4. We use spherical coordinates for storing selected 2D image positions to support rotation-invariant operations in the image space as described in Sec. 4.4. To convert an image space coordinate to spherical coordinates, we first back-project the pixel coordinate into the camera space as view ray as described in Eqn. A.24. Then, we apply the inverse kinematic operator $F : \mathbf{p} \to \mathbf{g}$ as defined in Eqn. 4.8, which maps the Euclidean 3D point $\mathbf{p}$ to the spherical point $\mathbf{g}$. Here, the distance for each back-project pixel is set to $d = 1$ meter[1]. Figure 5.3 shows the required steps for image-based selection and measuring of IP and for rendering rotation-invariant image points.

### 5.1.2 Support Point Method

To get the 3D coordinates of a building corner, the image pixel of the corner and a support point near the corner is defined, where the distance of the support point can be measured safely. Afterwards, the corner itself can simply be defined in the 2D image. The 3D coordinate of the target of interest is approximated by using the back-projected pixel of the first point and the measured distance of the support point. The approximation error becomes reasonable small for certain applications when following conditions hold: reasonable

[1]This approximation is valid for the simple spherical model of the RTS.

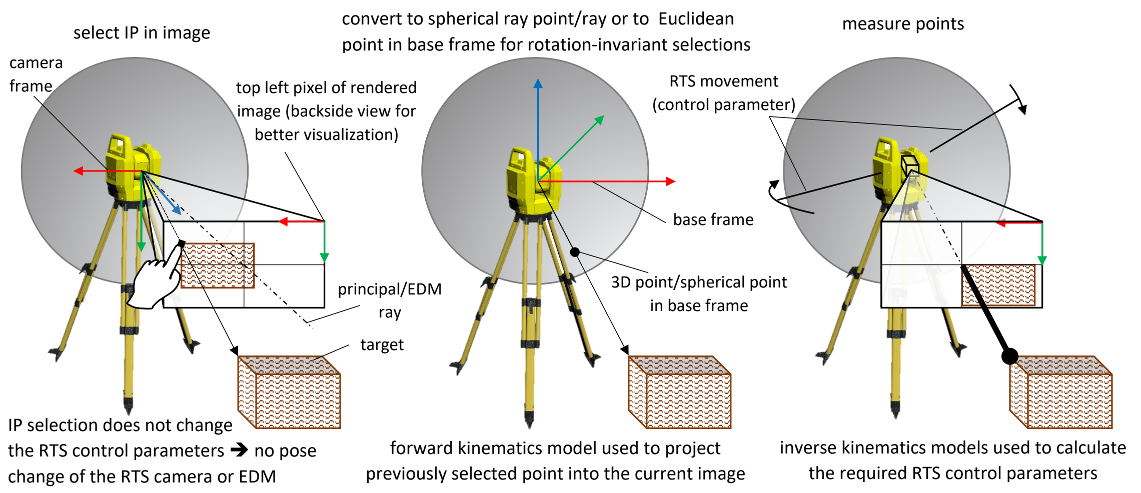

select IP in image
camera frame
top left pixel of rendered image (backside view for better visualization)
principal/EDM ray
target
IP selection does not change the RTS control parameters ➔ no pose change of the RTS camera or EDM
convert to spherical ray point/ray or to Euclidean point in base frame for rotation-invariant selections
RTS movement (control parameter)
base frame
3D point/spherical point in base frame
forward kinematics model used to project previously selected point into the current image
measure points
inverse kinematics models used to calculate the required RTS control parameters

Figure 5.3: Images-based selection and measurement of an IP. Rotation-invariant selections require the conversion of image points to an intermediate format. Depending on the underlying model, Spherical coordinates or DH control parameters can be used.

distance between the measurement device and the target, a perpendicular arrangement of the view ray and the measured surface, a small distance between the corner and the measured 3D point.

An offline version of this method is commonly used by surveying engineers [23, 60, 90]. With the support point method, the minimal measurement count for a 3D point is $N_{min} = 1$. Figure 5.2 shows the support point concept.

#### 5.1.2.1 Measurement Flow

The simple measurement flow is defined by following steps:

1. Use the pan/tilt control interface, until the target-of-interest is visible in the image
2. Define target-of-interest in the image
3. Define support point with a single distance measurement
4. Calculate the 3D position of the target-of-interest by using the angle of the image point and the distance of the support point measurement

#### 5.1.2.2 Calculating the Point

In the user interface, two 2D image points are defined: the pixel coordinates of the target-of-interest $\mathbf{u}_1$ and the pixel coordinates of the support point $\mathbf{u}_2$.

First both image points, $\mathbf{u}_1$ and $\mathbf{u}_2$, are converted to spherical coordinates using Eqn. 3.47

with distance $\mu = 1$ and Eqn. 4.8 to get the control values for the EDM pose of the RTS,

$$\mathbf{r}_1 = \begin{bmatrix} r_{1,x} \\ r_{1,y} \\ r_{1,z} \end{bmatrix} = \mathbf{X}(\mathbf{u}_1, 1) \qquad \mathbf{r}_2 = \begin{bmatrix} r_{2,x} \\ r_{2,y} \\ r_{2,z} \end{bmatrix} = \mathbf{X}(\mathbf{u}_2, 1) \qquad \begin{bmatrix} \varphi_1 \\ \theta_1 \\ 1 \end{bmatrix} = F(\mathbf{r}_1) \qquad \begin{bmatrix} \varphi_2 \\ \theta_2 \\ 1 \end{bmatrix} = F(\mathbf{r}_2) \tag{5.1}$$

where $\mathbf{r}_1$ and $\mathbf{r}_2$ defines a back-projected point at distance $\mu = 1$.

Then the distance $d_2$ is measured with the EDM at the angles $(\theta_2, \varphi_2)$ using the API of the RTS. Finally, we estimate the Euclidean 3D point $\mathbf{x}_1$ of the back-projected image point $\mathbf{u}_1$ using Eqn. 4.4 and the measured distance $d_2$:

$$\mathbf{x}_1 \approx G(\varphi_1, \theta_1, d_2) \tag{5.2}$$

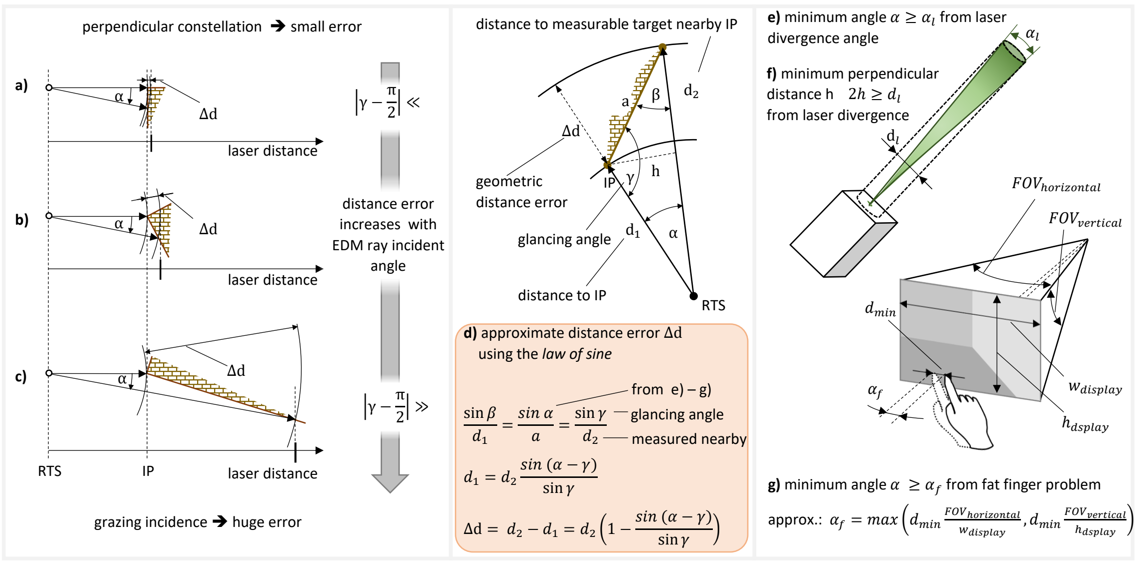

perpendicular constellation ➔ small error
a)
α
Δd
laser distance
$|\gamma - \frac{\pi}{2}| \ll$
b)
α
Δd
laser distance
distance error increases with EDM ray incident angle
c)
α
Δd
$|\gamma - \frac{\pi}{2}| \gg$
RTS
IP
laser distance
grazing incidence ➔ huge error
distance to measurable target nearby IP
a
β
d₂
Δd
geometric distance error
IP
γ
h
glancing angle
d₁
α
distance to IP
RTS
d) approximate distance error Δd using the law of sine
from e) – g)
$\frac{\sin\beta}{d_1} = \frac{\sin\alpha}{a} = \frac{\sin\gamma}{d_2}$
glancing angle
measured nearby
$d_1 = d_2 \frac{\sin(\alpha-\gamma)}{\sin\gamma}$
$\Delta d = d_2 - d_1 = d_2\left(1 - \frac{\sin(\alpha-\gamma)}{\sin\gamma}\right)$
e) minimum angle $\alpha \geq \alpha_l$ from laser divergence angle
$\alpha_l$
f) minimum perpendicular distance h $2h \geq d_l$ from laser divergence
$d_l$
$FOV_{horizontal}$
$FOV_{vertical}$
$d_{min}$
$\alpha_f$
$w_{display}$
$h_{dsplay}$
g) minimum angle $\alpha \geq \alpha_f$ from fat finger problem
approx.: $\alpha_f = max\left(d_{min}\frac{FOV_{horizontal}}{w_{display}}, d_{min}\frac{FOV_{vertical}}{h_{display}}\right)$

Figure 5.4: Systematic distance error approximation for measuring a horizontal wall using the support point method, reduced to 2D for reasons of simplicity. (a-c) An increasing incident angle leads to an increasing systematic distance error. (d) Assessment of the distance error as function of nearby point distance and incident angle. (e-f) Approximating the laser beam by a cylinder. (g) Considering the uncertainty of touch-screen based RTS targeting as function of the image resolution and the finger size.

### 5.1.2.3 Measurement Problem

While this method is commonly used in practice, it is critical for non-perpendicular measurement constellations as the systematic distance error increases with increasing incident angle between ray and target. Figure 5.4 shows an approximation of the systematic distance error for measuring a horizontal wall[2].

### 5.1.3 Support Line Method

Several 3D points on the visible wall are measured by the user to estimate a 3D line, which intersects the corner of interest. The corner itself can then simply be defined in the 2D image. The related 3D target is calculated by finding the intersection point of the back projected view ray with the previous estimated 3D line doing with a least square approximation.
With support lines, the minimal measurement count for 3D points is $N_{min} = 2$. When using more than two points, a robust estimation like RANSAC based least square 3D line fitting can be applied [37]. Figure 5.2 shows the support line concept.

#### 5.1.3.1 Measurement Flow

The simple measurement flow is defined by following steps:

1. Use the pan/tilt control interface, until the target-of-interest is visible in the image
2. Define target-of-interest in the image
3. Define support line with $N \geq 2$ measurements
4. Calculate the 3D position of the target-of-interest by intersecting the back-projected view ray with the support line

#### 5.1.3.2 Calculating the Support Line

For $N = 2$, the 3D support line can be written directly as Eqn. 5.7. Fitting the 3D line for $N > 2$ can be separated into two steps: fitting the 3D line position and fitting the 3D line direction. First, the center of mass of the 3D points is subtracted:

$$\mathbf{x}_i = \left[x_i, y_i, z_i\right]^T |_{i=0\ldots N-1} \qquad \bar{\mathbf{x}} = \frac{1}{N} \cdot \sum_{i=0}^{N-1} \mathbf{x}_i \qquad \mathbf{x}'_i = \mathbf{x}_i - \bar{\mathbf{x}}|_{i=0\ldots N-1} \tag{5.3}$$

with $\bar{\mathbf{x}}$ as center of mass of the 3D point set. The translated 3D points $\mathbf{x}'_i$ are now centered around 0. Then, the 3D points are normalized:

$$k = \max(|x'_{x,i}|, |x'_{y,i}|, |x'_{z,i}|)|_{i=0\ldots N-1} \qquad \mathbf{x}''_i = \frac{\mathbf{x}'_i}{k}|_{i=0\ldots N-1} \tag{5.4}$$

[2]We reduced the estimation to 2D for sake of clarity and for ease of comprehension.

and the 3D line orientation is calculated by stacking points and solving the maximization problem:

$$\mathbf{A}'' = \begin{bmatrix} x''_{x,0} & x''_{y,0} & x''_{z,0} \\ x''_{x,1} & x''_{y,1} & x''_{z,2} \\ \vdots & & \\ x''_{x,N-1} & x''_{y,N-1} & x''_{z,N-1} \end{bmatrix} \qquad \max_{||\mathbf{n}||=1} (||\mathbf{A}'' \cdot \mathbf{n}||) \tag{5.5}$$

The solution is the eigenvector that belongs to the largest eigenvalue and can be calculated by using SVD [62]. The line orientation is normalized for reasons of convenience:

$$\mathbf{n}' = \frac{\mathbf{n}}{||\mathbf{n}||} = \begin{bmatrix} n'_x \\ n'_y \\ n'_z \end{bmatrix} \tag{5.6}$$

A 3D line is fully specified by an arbitrary point on the line and the orientation. For consistent calculations, the 3D orientation can be interpreted as 3D direction vector. Using the center of mass $\bar{\mathbf{x}}$ and the normalized line direction $\mathbf{n}'$, the fitted 3D line $L$ in leased square sense is given by

$$L(t) = \bar{\mathbf{x}} + t \cdot \mathbf{n}' \tag{5.7}$$

#### 5.1.3.3 Intersecting the View Ray with the Support Line

First the 2D coordinate is back-projected to a 3D view ray using Eqn. 3.47. The best approximation for 3D line intersection can be found using Plücker coordinates [44]. However, we implemented 3D line intersection for two lines based on simple vector math [94].

### 5.1.4 Support Plane Method

To get the 3D coordinates of a building corner, the user measures several 3D points on the visible wall to estimate a planar approximation of this wall. The corner of interest can simply be defined in the 2D image. The related 3D target is calculated by intersecting the back-projected view ray with the previous estimated plane. The measurement concept is shown in Figure 5.2. The target-of-interest can be moved freely on the plane.

#### 5.1.4.1 Measurement Flow

The simple measurement flow is defined by following steps:

1. Use the pan/tilt control interface, until the target-of-interest is visible in the image
2. Define target-of-interest in the image
3. Define support plane with $N \geq 3$ measurements

4. Calculate the 3D position of the target-of-interest by intersecting the back-projected view ray with the support plane

#### 5.1.4.2 Calculating the Support Plane

In the easiest case the plane can be estimated by estimating the non-trivial solution of the linear homogeneous equation system

$$\mathbf{A} \cdot \mathbf{p} = \mathbf{0} \qquad \mathbf{A} = \begin{bmatrix} x_{x,0} & x_{y,0} & x_{z,0} & 1 \\ x_{x,1} & x_{y,1} & x_{z,2} & 1 \\ \vdots & & & \\ x_{x,N-1} & x_{y,N-1} & x_{z,N-1} & 1 \end{bmatrix} \tag{5.8}$$

where $\mathbf{A}$ is a matrix of stacked homogeneous 3D points with a 3D point count $N = 4$. The plane parameters $a$, $b$, $c$ and $d$ of the implicit plane equation are given by the $4 \times 1$ vector

$$\mathbf{p} = \begin{bmatrix} a & b & c & d \end{bmatrix}^T \qquad \mathbf{p}^T \cdot \begin{bmatrix} x_x & x_y & x_z & 1 \end{bmatrix}^T = \mathbf{0} \tag{5.9}$$

where $x_x$, $x_y$, $x_z$ are the coordinates of a 3D point on the plane.

Solving for $p$ in Eqn. 5.8 for $N \geq 4$ becomes a constrained least squares minimization problem

$$\min_{||\mathbf{p}||=1} (||\mathbf{A} \cdot \mathbf{p}||) \tag{5.10}$$

and can be solved with SVD [62].

A more robust plane estimation encounters some additional aspects:

- Minimal point set $N \geq 3$ instead of $N \geq 4$
- Normalization before computation for numerical stability
- RANSAC optimization for robustness against outliers in case of $N > 3$

For minimal point set, we must estimate the plane direction (rotation) and the plane translation separately. This procedure is analogous to the one for the support line, following Eqns. 5.3-5.6, but solving for the eigenvector that belongs to the smallest eigenvalue. Finally, the implicit plane representation is given by $\mathbf{p} = \begin{bmatrix} n'_x & n'_y & n'_z & -n'^T \cdot \bar{x} \end{bmatrix}^T$. This method requires at least $N_{min} = 3$ measured 3D points.

#### 5.1.4.3 Intersecting the View Ray with the Support Plane

First, the 2D coordinate is back-projected to a 3D view ray using Eqn. 3.47. The target-of-interest is given by the plane-ray intersection

$$t = \frac{-(\mathbf{n}'^T \cdot \mathbf{C} + d)}{n'^T \cdot \mathbf{n}_{ray}} \tag{5.11}$$

$$\mathbf{x}_{target} = \mathbf{C} + t \cdot \mathbf{n}_{ray} \tag{5.12}$$

with $n_{ray}$ as ray direction of the back projected image point, $C = 0$ as camera origin and $d = p(4)$ as distance between the origin and the intersection point [94]. If the denominator of Eqn. 5.11 is zero, the ray is either parallel to the plane or lies directly on the plane.

## 5.2 Simulation and Experiments

In this section, we describe the simulations and experiments undertaken. First, we describe the results of the MCS to analyze various aspects of the proposed measurement methods, such as influences of the target surfaces and the incident angles. Then, we shortly outline the experimental setup. Finally, we experimentally evaluate our methods in physical environments. Table 5.1 shows the test taxonomy for our MCS and experiments.

### 5.2.1 Monte-Carlo Simulations

For proper testing the methods described above, we use the RTS simulator as described in chapter 4. In particular, we defined a prototyping framework for MCS and for physical experiments as shown in Fig. 5.5. The abstraction layer on top of the RTS API allows for seamless exchange of simulator and physical device. For simulation, the prototyping framework is set up to carry out the MCS with the real-time RTS simulator. The test sets for the MCS are generated in MATLAB, control values and simulation parameters are uploaded to Unity3D; the measurements are simulated in Unity3D, results are streamed back and are evaluated in MATLAB.

The simulation setup inherently provides ground truth and a common coordinate frame for all measurements, devices and targets. This allows for easier comparison of the different methods.

Figure 5.6 shows the simulator and the MCS workflow.

#### 5.2.1.1 Measurement Targets Variants

The basic target is a planar triangle mesh, placed at ten meter distance from the RTS. Different target variants are generated using following steps: 1. Subdivide the surface of the basic target into small triangles, 2. translate the mesh vertices, and 3. remove faces and vertices outside of the region of interest (ROI) for performance reasons.

Table 5.1: Test taxonomy. (x) evaluated; (x*) evaluated, where the parameter approximately fits the specification; (-) not evaluated or not applicable.

| test configurations | | test environment | | | |
|---|---|---|---|---|---|
| property | value | MCS | laboratory | indoor | outdoor |
| surface type | planar | x | x* | x* | x* |
| | uneven | x | - | - | - |
| | fillet | x | - | - | - |
| incident angle $\alpha$ | $0.5\pi$ | x | x | x | x |
| | $0.25\pi$ | x | x | x | x |
| measurement method | direct | x | x | x | x |
| | nearby | x | x | x | x |
| | support point | x | x | x | x |
| | support line | x | x | x | x |
| | support plane | x | x | x | x |
| noise | n1 (no noise) | x | - | - | - |
| | n2 (EDM noise) | x | - | - | - |
| | physical | - | x | x | x |
| evaluation method | direct: $\mathbf{x}_{i,IP,ref} - \mathbf{x}_{i,IP,est}$ | x | - | - | - |
| | indirect: $\mathbf{x}_{i,IP,est} - \mathbf{x}_{i+1,IP,est}$ | - | x | x | x |

We simulate three different target variants with following surface properties: 1. planar surface, 2. uneven surface, and 3. round edges (fillet). The planar surface variant is simply the basic target. The uneven surface variant is generated using random translations of the mesh vertices along the vertex normals. Similar, the fillet of the target with round edges is generated by translating the vertices near the border as a function of the distance to the border. Figure 5.7 shows the generation of the mesh variants.

#### 5.2.1.2 RTS Sensor Uncertainty Simulation

We follow the JCGM 100:2008 GUM [57] for modeling the sensor uncertainty. In particular, GUM standardizes the analysis and report of measurement uncertainties of measured physical quantities to allow repeatable experiments. The uncertainty of RTS sensors with normal distributed random noise can be specified in following general form:

$$p(|y - x| \leq k u_c(y)) = CI_k \tag{5.13}$$

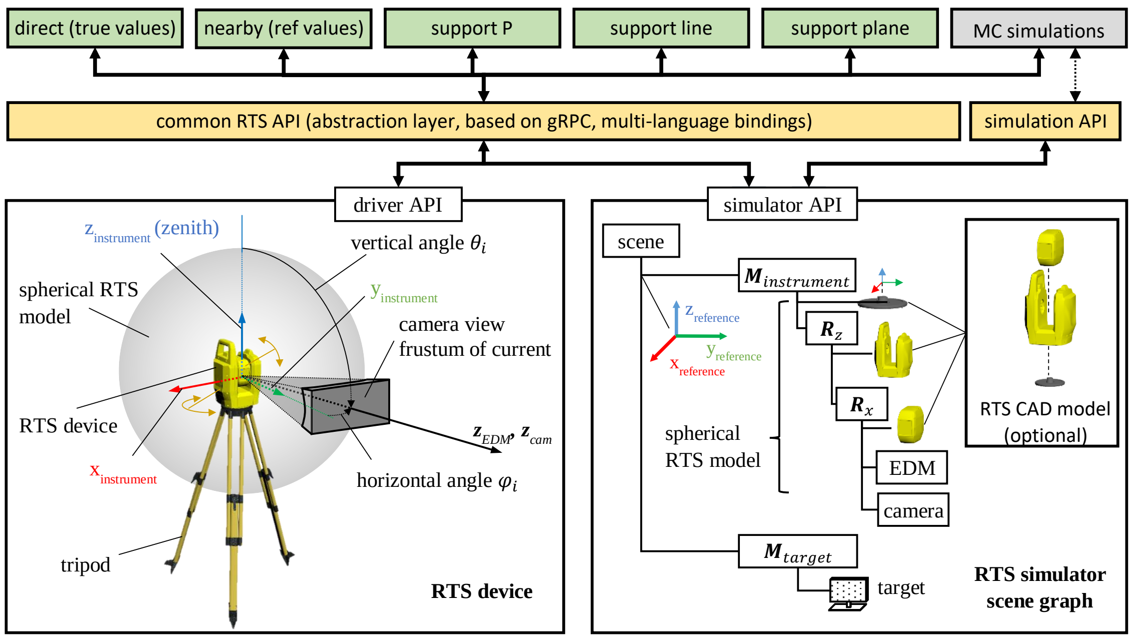


Figure 5.5: RTS prototyping framework, here used for MCS and physical experiments. The abstraction layer, which provides a common API for the RTS hardware and simulator, was generated automatically with support of the gRPC library. Here, we used the simplified geometric RTS model in Unity3D. Additional API methods allow modifications of the scene graph and various simulation properties.

where $x$ is the measured quantity, $u_c(y)$ is the combined standard uncertainty of the measurement result $y$; $k$ is the coverage factor, and $CI_k$ is the confidence interval[3]. Let $u_a(y)$ be an additive and $u_p(y)$ be a proportional component of the combined sensor uncertainty, both provided by the device manufacturers. Then, $u_c(y)$ is given by

$$u_c(y) \approx \sqrt{u_a(y)^2 + (xu_p(y))^2} \tag{5.14}$$

Unity3D provides generators for uniform distributed random values. We use the Box-Muller transform [18] to simulate normal distributed noise for sensor readings:

$$y = x + \sqrt{-2ln(g)} \cos\left(2\pi h u_c(y)\right) \tag{5.15}$$

where $u_c(y)$ is the desired standard uncertainty, $(g, h)$ are uniformly distributed random values, and $x$ is the simulated sensor reading without noise. The EDM uncertainty has significant influence on the measurements and should be analyzed. The angle uncertainty of actuators is negligibly small and therefore is not considered in the calculations. Table 5.2 provides the sensor uncertainty settings for the MCS, Fig. 5.8 shows the noise simulation

[3]Analogously to GUM, we use the same symbol is as the physical quantity and as the random variable for economy of notation [57].

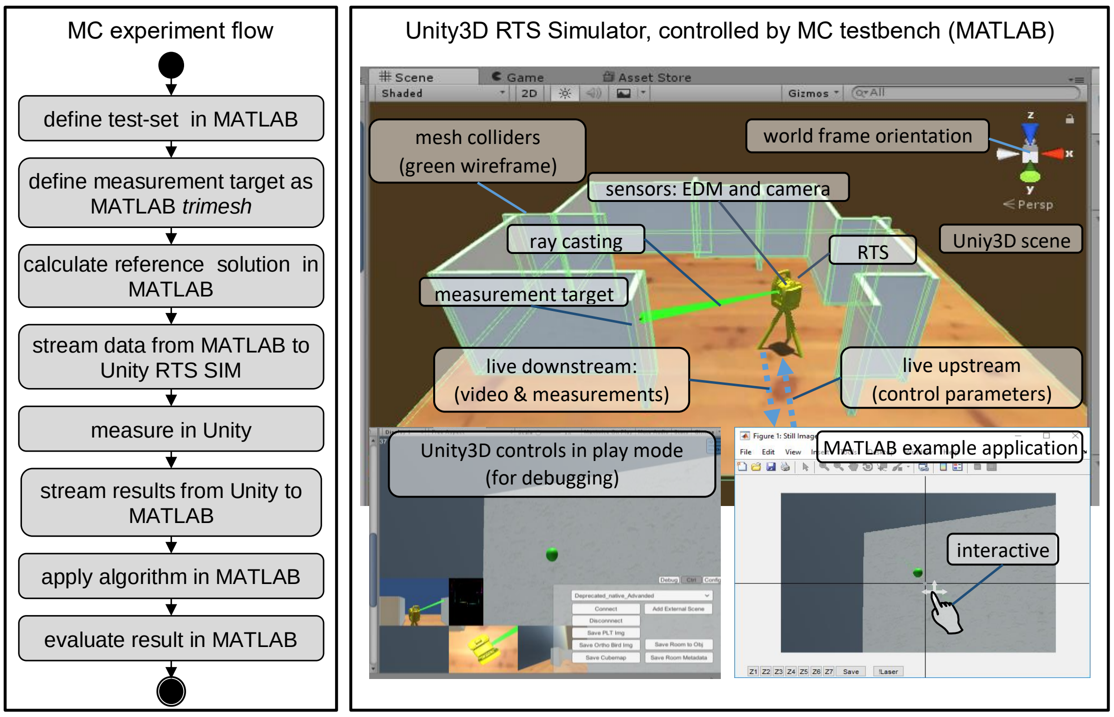

MC experiment flow
define test-set in MATLAB
define measurement target as MATLAB *trimesh*
calculate reference solution in MATLAB
stream data from MATLAB to Unity RTS SIM
measure in Unity
stream results from Unity to MATLAB
apply algorithm in MATLAB
evaluate result in MATLAB
Unity3D RTS Simulator, controlled by MC testbench (MATLAB)
mesh colliders (green wireframe)
world frame orientation
sensors: EDM and camera
ray casting
RTS
Uniy3D scene
measurement target
live downstream: (video & measurements)
live upstream (control parameters)
Unity3D controls in play mode (for debugging)
MATLAB example application
interactive

Figure 5.6: (left) MCS flow. (right) RTS Unity3D simulator.

architecture. More general error descriptions can include signal refraction, cyclic errors, pointing errors and camera calibration effects, but are beyond the scope of this work [106].

Table 5.2: Sensor noise settings for MCS.

| Description | | EDM Sensor | | Angle Sensor | |
|---|---|---|---|---|---|
| Label | Description | $u_a(d)$ | $u_p(d)$ | $u_a(\alpha)$ | $u_p(\alpha)$ |
| n1 | without noise | 0 | 0 | 0 | 0 |
| n2 | with noise | $0.75e-3$m | $10e-6$m | 0 | 0 |

### 5.2.1.3 Complex Collider Definition for Ray Casting

Unity3D allows using triangle meshes as colliders for physical simulations [105]. The close coupling with the GPU limits the numeric precision of scene operations to 32 bit floating-point arithmetic[4]. In general, a higher precision is not required for the proposed MCS. However, the non-convex measurement targets require non-convex colliders, which cause ray casting problems due to numeric round-off effects. Figure 5.9 shows a ray casting experiment where the ray simply passes through a surface when targeting a mesh vertex or

[4]Higher precision arithmetic require explicit implementation of the scene graph and related operations.

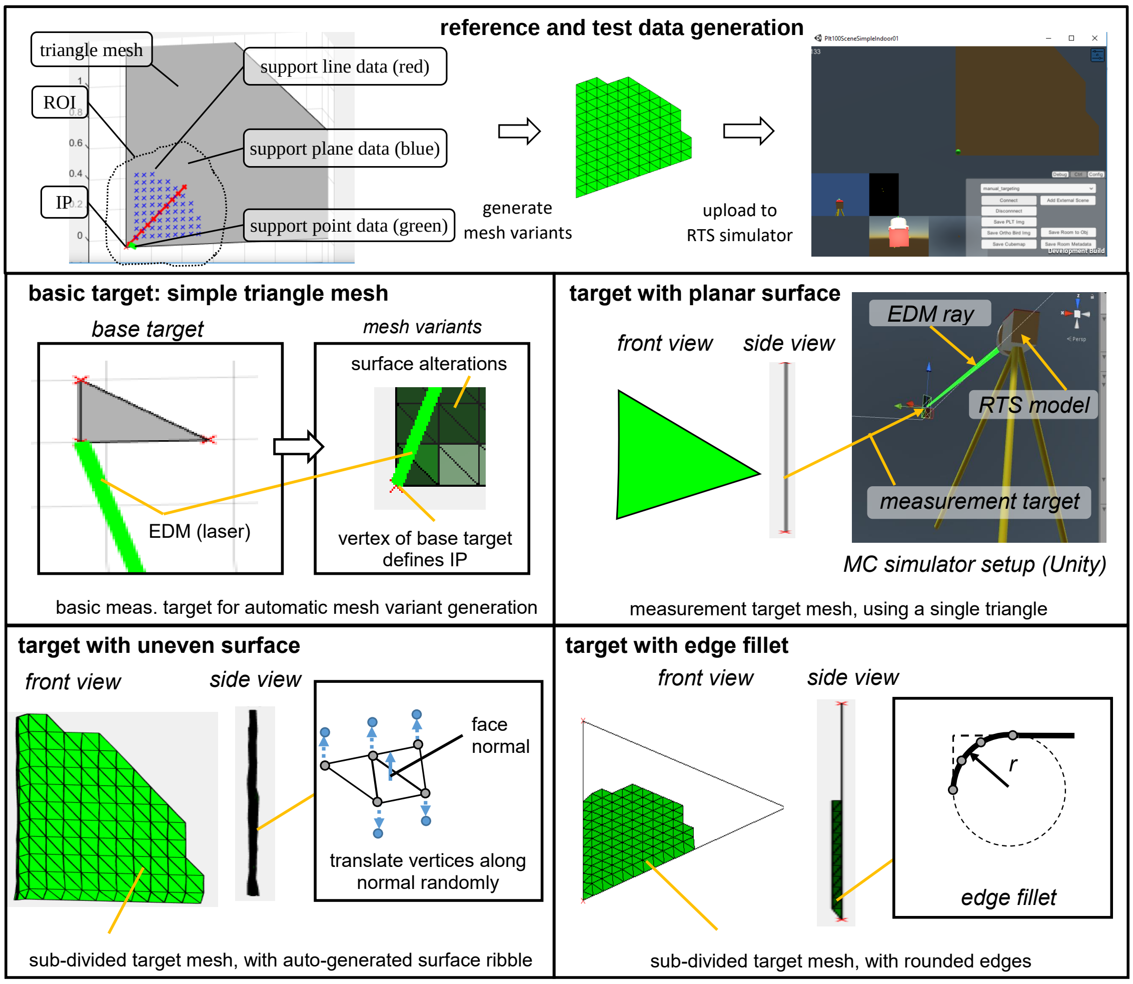

reference and test data generation
triangle mesh
ROI
IP
support line data (red)
support plane data (blue)
support point data (green)
generate mesh variants
upload to RTS simulator
basic target: simple triangle mesh
base target
mesh variants
surface alterations
EDM (laser)
vertex of base target defines IP
basic meas. target for automatic mesh variant generation
target with planar surface
front view
side view
EDM ray
RTS model
measurement target
MC simulator setup (Unity)
measurement target mesh, using a single triangle
target with uneven surface
front view
side view
face normal
translate vertices along normal randomly
sub-divided target mesh, with auto-generated surface ribble
target with edge fillet
front view
side view
r
edge fillet
sub-divided target mesh, with rounded edges

Figure 5.7: Generation of targets for MCS.

edge directly. This is critical for our experiments, thus explicit colliders must be generated. We simply increase each triangle of the target surface by $0.5e-4m$. In particular, we perform the following steps: 1. First, we remove the links between connected triangles by duplicating shared vertices. 2. Then, we translate the vertices of a triangle along the medians, the line between a vertex and the centroid, to enlarge the surface. While this method decreases the simulation accuracy, it also increases the reliability of the ray casting. The generated colliders consist of overlapping triangles, and they counteract intersection issues caused by round-off errors.

### 5.2.1.4 Additional MCS parameters

The main parameters for the MCS are defined in Table 5.1, sensor noise parameters are given in Table 5.2. Additional settings are required for defining the MCS, such as measurement count for each method, distance between the IP point and the measured point

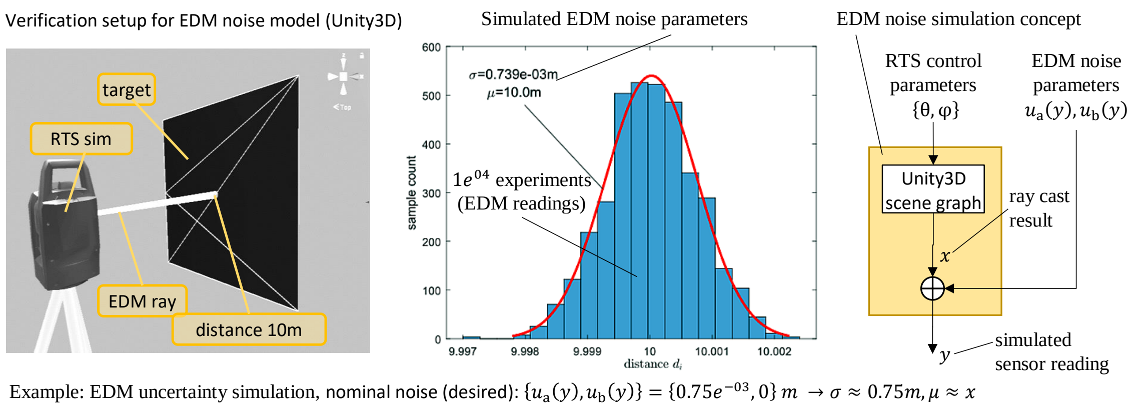

Verification setup for EDM noise model (Unity3D)
target
RTS sim
EDM ray
distance 10m
Simulated EDM noise parameters
σ=0.739e-03m
μ=10.0m
1e04 experiments
(EDM readings)
sample count
distance d_i
EDM noise simulation concept
RTS control parameters {θ, φ}
EDM noise parameters u_a(y), u_b(y)
Unity3D scene graph
ray cast result
x
y
simulated sensor reading
Example: EDM uncertainty simulation, nominal noise (desired): {u_a(y), u_b(y)} = {0.75e^-03, 0} m → σ ≈ 0.75m, μ ≈ x

Figure 5.8: Uncertainty simulation for RTS sensors.

for the nearby method, and properties for surface variant generation. Table 5.3 lists the additional MCS properties, which we used for this work.

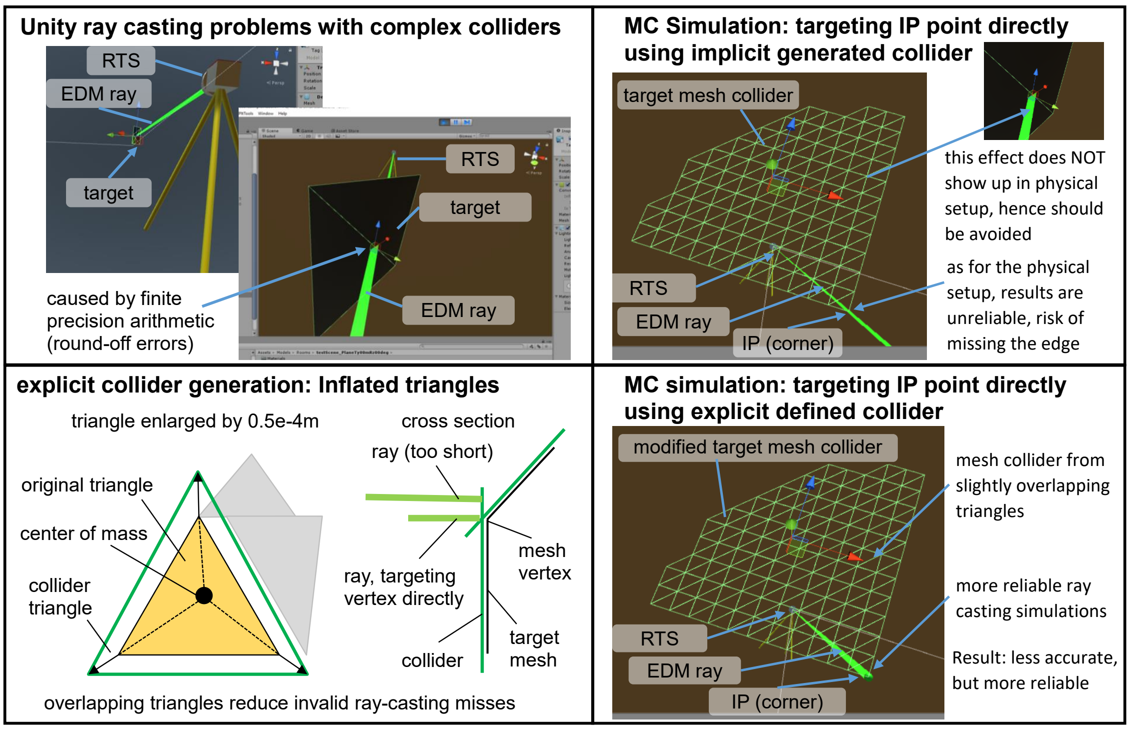

Unity ray casting problems with complex colliders
RTS
EDM ray
target
caused by finite precision arithmetic (round-off errors)
MC Simulation: targeting IP point directly using implicit generated collider
target mesh collider
this effect does NOT show up in physical setup, hence should be avoided
as for the physical setup, results are unreliable, risk of missing the edge
IP (corner)
explicit collider generation: Inflated triangles
triangle enlarged by 0.5e-4m
original triangle
center of mass
collider triangle
cross section
ray (too short)
ray, targeting vertex directly
mesh vertex
target mesh
collider
overlapping triangles reduce invalid ray-casting misses
MC simulation: targeting IP point directly using explicit defined collider
modified target mesh collider
mesh collider from slightly overlapping triangles
more reliable ray casting simulations
Result: less accurate, but more reliable

Figure 5.9: Explicit collider generation for Unity3D to avoid ray casting issues of complex colliders.

Table 5.3: Additional parameters for the MCS used in this work.

| property | value |
|---|---|
| bounding box for basic target | 2m |
| ROI radius | 0.5m (region for picking additional points) |
| subdivision iterations | fillet target: 50; other targets: 25 |
| fillet surface jitter | 10e-3m |
| fillet radius | 30e-3m |
| test count per MCS | 100 |
| distance between RTS and target | 10m |
| inflate vertex offset for colliders | 0.5e-4m |
| RANSAC line/plane fitting | no |
| minimum safety distance between ray and target edges for non-direct methods | $2.5e-3$m (circular approximation of the projected EDM ray at the intersection point, assuming 5e-3m radius) |

#### 5.2.1.5 Results

Table 5.4 shows the results of the MCS for all 60 variants. The direct method is used to estimate reference values; the nearby method is considered as standard method when no additional corrections are applied. Figure 5.10 shows the box-and-whisker plots for the simulations with applied EDM sensor noise. The plots visualize following *robust summary statistics*[5]: 1. The central mark is the median, 2. the bottom and top box boundaries are the 25th and 75th percentiles, respectively; 3. the + symbols show the outliers, and 4. the whiskers show the most extreme inlier data points.

### 5.2.2 Experiments

We further performed several experiments in laboratory and outdoor environments. The registration of different measurement sets and the measurement targets in a common coordinate system relies on the measurement of control points. However, the point measurement methods themselves are the subject of the current analysis. Alternative registration methods use a fixed installation of reflective targets. In this work, we do not register the measured point sets in a common frame to avoid the physical installation.

We applied an indirect analysis of the measurement error instead, which did not require a common coordinate frame for the measurement sets. In particular, we measured the distance between two corners of a flat surface, whereby only the front face of the surface was fully visible.

This simple evaluation setup does not require any special measurement equipment and

[5]MATLAB standard settings for box plots, function *boxplot*, *statistics toolbox*.

Table 5.4: MCS results. The direct method is usually not applicable for physical corner targets without target modifications. The traditional approach that we use for comparison is the nearby method, which is marked in red.

| MCS settings | | | EDM noise: $u_a(d) = 0m$, $u_p(d) = 0m$ | | | EDM noise: $u_a(d) = 0.75e-3m$, $u_p(d) = 10e-6m$ | | |
|---|---|---|---|---|---|---|---|---|
| $\alpha$ | mesh variant | meas. method | $E(\mathbf{x}_{ip,ref} - \mathbf{x}_{ip,est})$ [m] | $\bar{d}$ [m] | $\sigma(d)$ [m] | $E(\mathbf{x}_{ip,ref} - \mathbf{x}_{ip,est})$ [m] | $\bar{d}$ [m] | $\sigma(d)$ [m] |
| $0.5\pi$ | planar | direct | $1.961e-06$ | $1.961e-06$ | $4.257e-21$ | $6.076e-04$ | $6.076e-04$ | $4.114e-04$ |
| | | nearby | $2.500e-03$ | $2.500e-03$ | $4.359e-18$ | $2.595e-03$ | $2.595e-03$ | $\mathbf{1.154e-04}$ |
| | | support point | $2.500e-03$ | $7.052e-05$ | $9.535e-20$ | $2.593e-03$ | $5.652e-04$ | $4.201e-04$ |
| | | support line | $2.525e-01$ | $\mathbf{1.336e-06}$ | $\mathbf{6.385e-22}$ | $2.525e-01$ | $\mathbf{5.137e-04}$ | $3.732e-04$ |
| | | support plane | $3.079e-01$ | $1.527e-06$ | $1.915e-21$ | $3.071e-01$ | $1.277e-03$ | $9.642e-04$ |
| | uneven | direct | $7.668e-07$ | $\mathbf{7.668e-07}$ | $\mathbf{1.064e-22}$ | $6.260e-04$ | $\mathbf{6.260e-04}$ | $4.689e-04$ |
| | | nearby | $2.512e-03$ | $2.512e-03$ | $5.666e-18$ | $2.597e-03$ | $2.597e-03$ | $\mathbf{1.419e-04}$ |
| | | support point | $2.545e-03$ | $4.770e-04$ | $9.807e-19$ | $2.629e-03$ | $6.852e-04$ | $4.687e-04$ |
| | | support line | $3.268e-01$ | $5.816e-03$ | $9.589e-18$ | $2.249e-01$ | $5.971e-03$ | $5.282e-04$ |
| | | support plane | $3.280e-01$ | $5.011e-03$ | $5.230e-18$ | $3.713e-01$ | $8.460e-03$ | $1.158e-03$ |
| | fillet | direct | $5.338e-08$ | $\mathbf{5.338e-08}$ | $\mathbf{1.197e-22}$ | $6.112e-04$ | $\mathbf{6.112e-04}$ | $4.774e-04$ |
| | | nearby | $2.495e-03$ | $2.495e-03$ | $3.051e-18$ | $2.608e-03$ | $2.608e-03$ | $\mathbf{1.895e-04}$ |
| | | support point | $2.495e-03$ | $7.243e-05$ | $5.448e-20$ | $2.605e-03$ | $6.148e-04$ | $4.616e-04$ |
| | | support line | $2.637e-01$ | $1.599e-02$ | $1.395e-17$ | $2.815e-01$ | $1.320e-02$ | $5.923e-04$ |
| | | support plane | $3.044e-01$ | $3.003e-02$ | $6.974e-17$ | $3.446e-01$ | $1.055e-02$ | $1.020e-03$ |
| $0.25\pi$ | planar | direct | $1.527e-07$ | $1.527e-07$ | $\mathbf{0}$ | $5.453e-04$ | $5.453e-04$ | $4.290e-04$ |
| | | nearby | $2.500e-03$ | $2.500e-03$ | $1.743e-18$ | $2.580e-03$ | $2.580e-03$ | $\mathbf{3.385e-04}$ |
| | | support point | $2.500e-03$ | $1.250e-03$ | $\mathbf{0}$ | $2.613e-03$ | $1.337e-03$ | $7.076e-04$ |
| | | support line | $2.618e-01$ | $2.770e-07$ | $\mathbf{0}$ | $2.247e-01$ | $\mathbf{5.306e-04}$ | $4.124e-04$ |
| | | support plane | $3.325e-01$ | $\mathbf{6.443e-09}$ | $4.988e-24$ | $2.993e-01$ | $7.783e-04$ | $5.983e-04$ |
| | uneven | direct | $1.937e-07$ | $\mathbf{1.937e-07}$ | $\mathbf{1.862e-22}$ | $6.223e-04$ | $\mathbf{6.223e-04}$ | $\mathbf{4.588e-04}$ |
| | | nearby | $2.491e-03$ | $2.491e-03$ | $5.230e-18$ | $2.728e-03$ | $2.728e-03$ | $4.731e-04$ |
| | | support point | $2.813e-03$ | $1.799e-03$ | $2.179e-18$ | $2.773e-03$ | $1.640e-03$ | $7.980e-04$ |
| | | support line | $1.584e-01$ | $1.273e-03$ | $1.308e-18$ | $2.696e-01$ | $2.950e-03$ | $5.019e-04$ |
| | | support plane | $3.243e-01$ | $4.674e-04$ | $4.359e-19$ | $4.219e-01$ | $2.874e-02$ | $1.933e-03$ |
| | fillet | direct | $1.527e-07$ | $\mathbf{1.527e-07}$ | $\mathbf{0}$ | $6.047e-04$ | $6.047e-04$ | $4.769e-04$ |
| | | nearby | $2.575e-03$ | $2.575e-03$ | $6.974e-18$ | $2.650e-03$ | $2.650e-03$ | $4.088e-04$ |
| | | support point | $2.575e-03$ | $1.326e-03$ | $\mathbf{0}$ | $2.669e-03$ | $1.361e-03$ | $7.446e-04$ |
| | | support line | $1.777e-01$ | $1.565e-02$ | $\mathbf{0}$ | $2.410e-01$ | $4.244e-02$ | $8.440e-04$ |
| | | support plane | $3.217e-01$ | $2.919e-02$ | $\mathbf{0}$ | $3.055e-01$ | $1.936e-02$ | $9.770e-04$ |

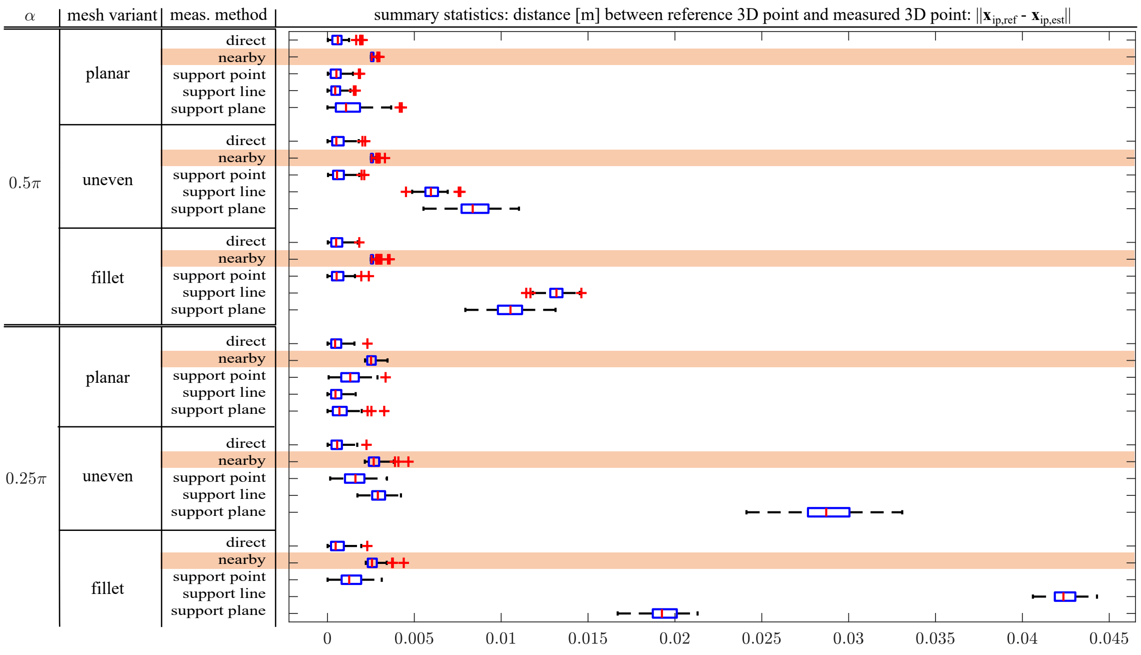


Figure 5.10: Box-and-whisker plot of the MCS results, with applied EDM noise. The direct method is usually not applicable for physical corner targets without target modifications. The traditional approach that we use for comparison is the nearby method, which is marked in red.

therefore can be easily be applied in controlled indoor and in selected outdoor environments. The results are also not influenced by the registration uncertainty of the RTS, which increases the repeatability and reproducibility of the proposed experiments. While this method is sufficient for comparative studies for non-direct measurable targets, it does not follow the ISO 17123 standard [52].

For physical experiments and interactive tests, we designed a graphical user interface (GUI) that allows seamlessly interfacing the RTS simulator or the physical RTS device. The GUI provides intuitive interactions with the implemented workflows and allows even novice and non-expert users to use the proposed measuring methods within a few minutes.

For each test, the user selects a particular measurement method. After selection of the method, the operator is automatically guided through the process to fulfill the measuring task, with a final result given at the end.

The RTS for our experiments had been fully calibrated by the manufacturer. The driver provides access to sensors and actuators of the device and transforms sensor data between the different coordinate systems; sensor data corrections are applied internally.

The GUI shown in Figure 5.11 is used for convenient access of the implementation and for allowing even novice and non-expert users to use the methods in an intuitive way. Training time to introduce the concepts of measuring and the individual methods was thereby reduced to only around 10 minutes. After selection of the given method, the operator is automatically guided through the process to fulfill the measuring task, with a final result given at the end.

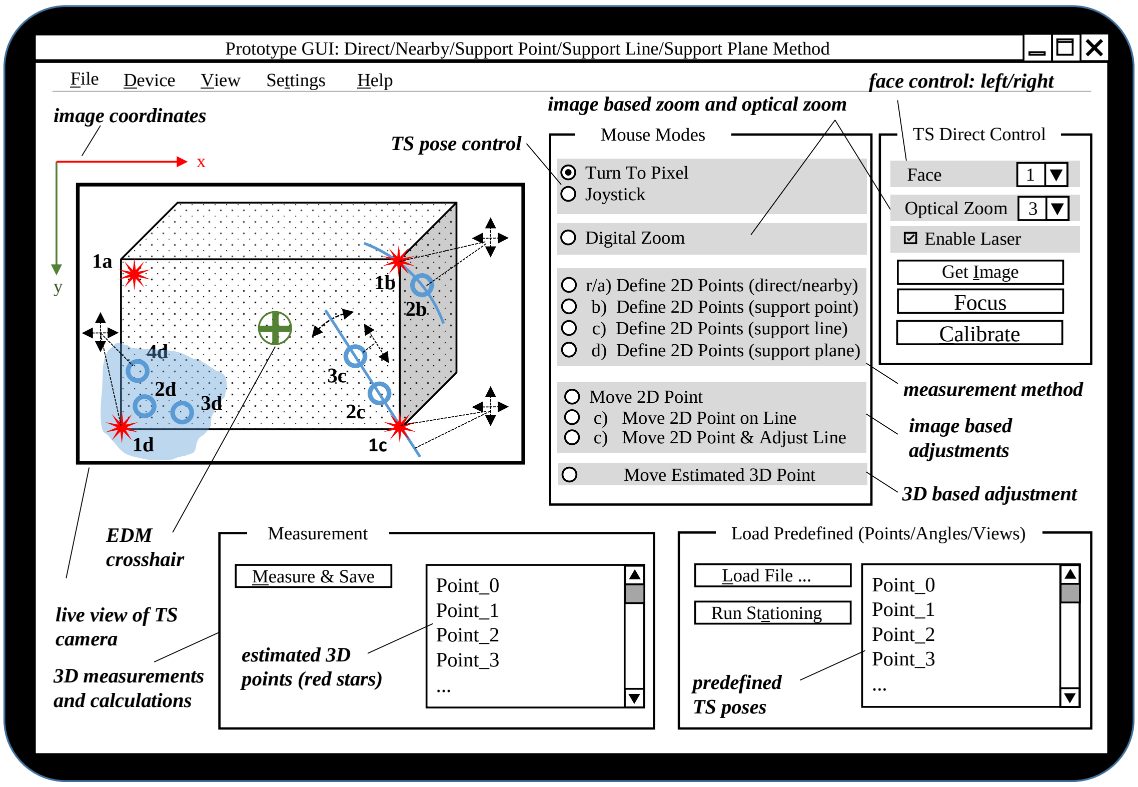


Figure 5.11: Test GUI used in our system [65]. The interface guides the user through the measurement tasks.

For ground truth measurement, the RTS and the target are positioning appropriately as follows:

- Approximately same height of target center and camera center
- Approximately perpendicular laser beam direction for laboratory experiments and outdoors for ground truth measurements
- Approximately perpendicular laser beam direction for ground truth measurements and $0.25\pi$ direction for outdoor evaluation

The setup is shown in Figure 5.12. The distance between the measurement target and the RTS is about 5m for all experiments. The distance between the two top corners of the measurement indoor target is about 0.6m.

The discussion in the following refers to the physical measurement results, which are given in Table 5.5.

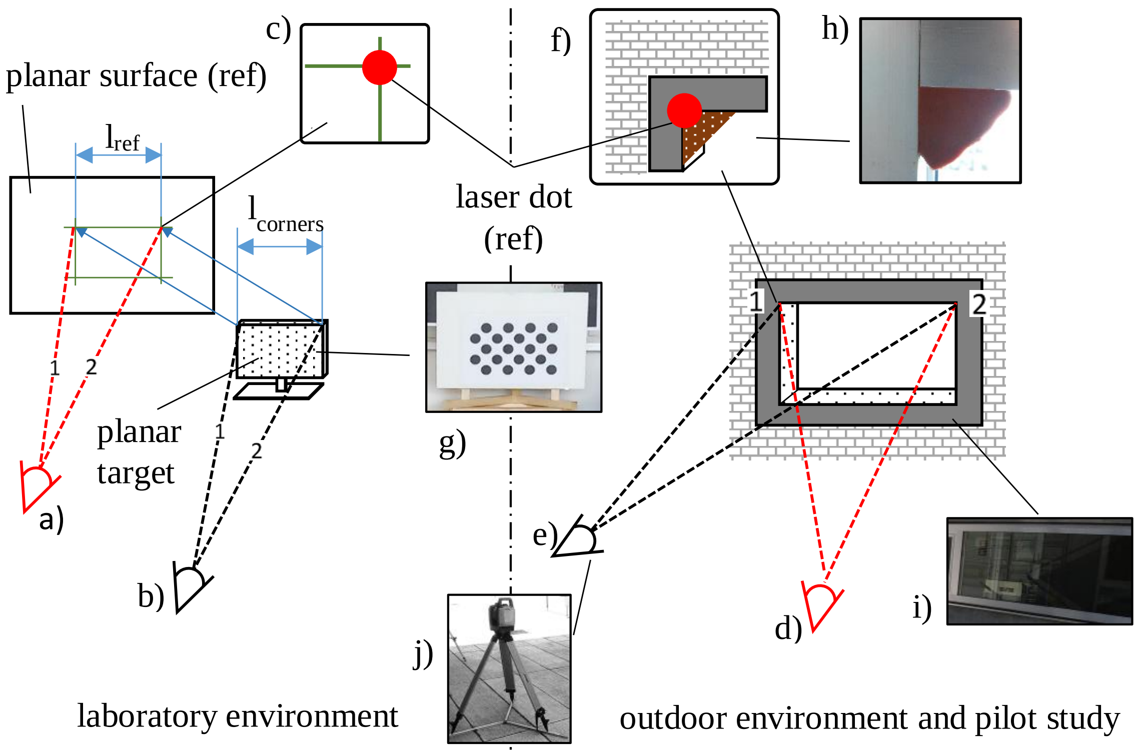


Figure 5.12: Measurement setups for testing under laboratory conditions and for outdoor scenarios [65]: a) measurement of the reference distance between the two top corners of the portable target, b) portable target used to measure the distance between two corners in laboratory conditions, c) detailed view of the projected laser dot during the reference measurement, d) reference measurement of a window in indoor and outdoor conditions using perpendicular viewing angle, e) the same windows measured with a viewing angle of $0.25\pi$, f) and h) the modeling clay for reference measurements, i), g) and j) the outdoor window, the portable laboratory target and the RTS.

#### 5.2.2.1 Measurement Strategy

For Euclidean distance evaluation, a single set measurement consists of the measured 3D position of the first and the second corner of the target[6]. All measurements where converted to Euclidean coordinates using the API of the device driver. The result is given in the confidence interval of $\pm 2\hat{\sigma}_d$, with $\hat{\sigma}_d$ as unbiased standard deviation assuming unbiased normal distribution of the measurements:

[6]Note that we use a half-set for our evaluations, since we do not use the second telescope face (face right).

The Euclidean distance of measurement $i$ between two points $p_{i,0}$ and $p_{i,1}$ is calculated by

$$d_i = ||\mathbf{p}_{i,1} - \mathbf{p}_{i,0}|| = || \begin{bmatrix} x_{i,1} \\ y_{i,1} \\ z_{i,1} \end{bmatrix} - \begin{bmatrix} x_{i,0} \\ y_{i,0} \\ z_{i,0} \end{bmatrix} || \tag{5.16}$$

and the average distance $\bar{d}$ and the unbiased standard deviation $\hat{\sigma}$is given by

$$\bar{d} = \frac{\sum_{i=0}^{N-1} d_i}{N} \qquad \hat{\sigma} = \sqrt{\frac{\sum_{i=0}^{N-1} (d_i - \bar{d})^2}{N-1}} \tag{5.17}$$

For outlier removal, at least $N = 3$ sets must be measured. Outliers are removed using median absolute deviation (MAD) with $\pm 3\hat{\sigma}$ interval on distances [71]. The statistic evaluation is repeated on the reduced data set.

We calculate the distance error between two points $d$ using

$$\Delta d = |\bar{d}_{ref} - \bar{d}| \pm 2 \cdot \sqrt{\hat{\sigma_{ref}}^2 + \hat{\sigma_1}^2} \tag{5.18}$$

with $\bar{d}_{ref} \pm 2\hat{\sigma}_{dref}$ as reference distance and $\bar{d} \pm 2\hat{\sigma}$ as measured distances between two corners.

For measuring the ground truth, we employed two different approaches. For the laboratory target, we aligned it with a planar surface and measured the distance using the RTS. Note that this method is suitable for portable targets and outer corners only. For ground truth estimation of immovable targets like windows, we filled the corners with modeling clay to create a quasi-planar surface around the corners, which could be measured by the RTS. This method is suitable for fixed and portable targets and is well suited for inner corners[7].

**Measurement Setup** We measured the distance between two corners of a flat surface, whereby only the front face of the surface is fully visible. This is achieved by appropriately positioning the target and the RTS:

- Approx. same height of target center and camera center
- Approx. perpendicular laser beam direction for laboratory experiments and outdoors for ground truth measurements
- Approx. perpendicular laser beam direction for ground truth measurements and 45° direction for outdoor evaluation

The measurement setup is shown in Figure 5.12. The distance between the RTS and the measurement target is about 5m in all experiments. The distance between the two top corners of the measurement indoor target is about 0.6m.

[7]Note that we performed the ground truth measurements immediately before the experiments, to ensure that errors due to changes in environmental conditions are negligible.

**Test Hardware and Limitations** The RTS we used for our experiments had been fully calibrated by the manufacturer. It provides a closed source driver for controlling the device, for retrieving the camera image and for translating between all coordinate systems. As common for commercially available systems, there is no direct access to the raw data of the sensors, the geometric model and the related parameters. The applied correction algorithms are confidential and kept secret by manufacturers. While in our case the available API does not provide a projection matrix, it offers a complete API for coordinate system conversions. Therefore, we use the simplified geometric model, which is shown in Figure 4.1, for a calibrated RTS to explain our methods.

Note that whether or not it is theoretically possible to perform full manual calibration for a single instance of a device at hand, it is neither reasonable to assume that every device is shipped with a calibration by the manufacturer that is perfect for each possible measurement situation, nor is manual calibration easily possible given the level of access provided by APIs. Thus our assumption to work with the stock calibrated device *as is* and employing our simplified geometric model is plausible.

#### 5.2.2.2 Laboratory Measurements

We conducted two experiments with the portable target in laboratory conditions. First, we measured the ground truth distance between the two top corners, as shown in Fig. 5.12 a) and b). Then we used the four different methods to perform the measurement again, where we used a laser incident angle of $\alpha \approx 90°$. The support line and support plane methods either outperform the others or perform on par, as shown in Table 5.5.

In a second experiment, we measured the same distance again with the total station, but pointing at the target with a laser incident angle of $\alpha \approx 45°$. Again, the results indicate that the support line and support plane based methods achieve considerably better results than the standard method and the support point method.

#### 5.2.2.3 Indoor Measurements

We performed two experiments with the EDM laser incident angles $\alpha \approx \{90°, 45°\}$, respectively, using a window as seen from the interior of a building. Here, we measured the distance between two corners of the window as shown in Figure 5.12 d) and e). Overall, the support line and support plane based methods achieve considerably better results than the standard method and the support point method, or perform at least on par.

#### 5.2.2.4 Outdoor Measurements

We conducted four outdoor experiments, where we measured the extents of a window from a perpendicular and a 45° point of view. The particular hardware setup is shown in Figure 5.13. First, we measured the ground truth distance as shown in Figure 5.12 d), f) and h). Then, we applied the four measurement methods again. The support line and the support

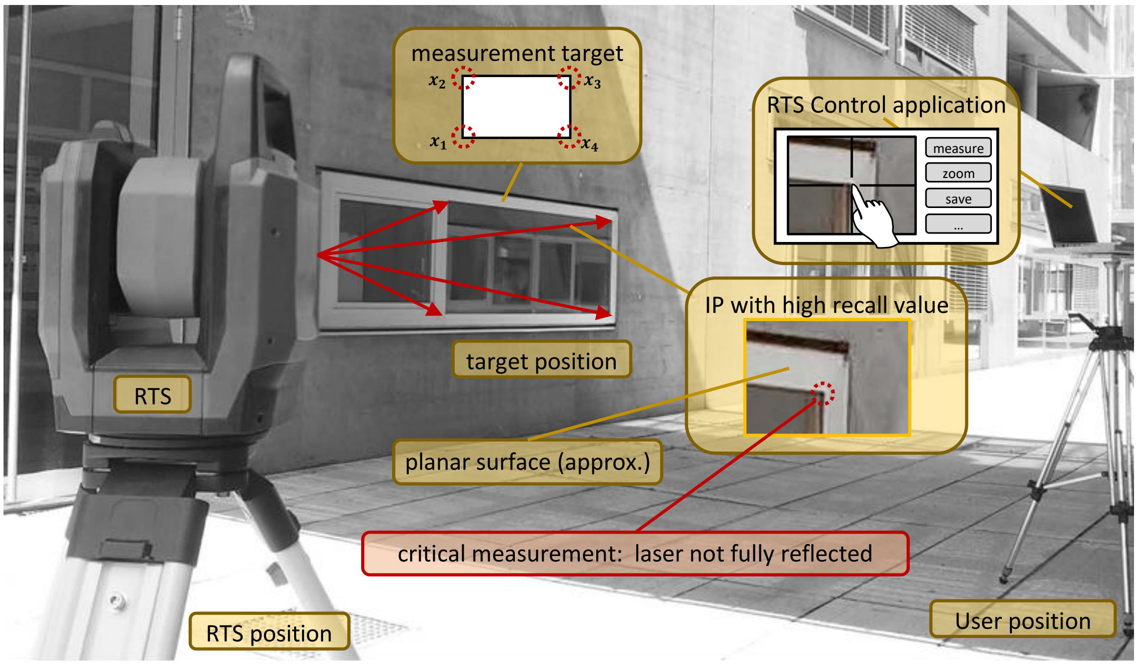


Figure 5.13: Measurement setup for our experiments [65]. The RTS is placed on two different positions for testing the influence of the laser incident angle. The user controls the RTS remotely using a vision-based prototyping software on the mobile PC. The laser is barely visible from close distance, but not from the user position or in the live camera stream. RTS and mobile PC used in our experiments. The communication between the devices is done over Wi-Fi.

plane methods are overall more suitable and give better results, or perform at least on par, as shown in Table 5.5.

#### 5.2.2.5 Pilot Study

In analogy to the experiment described above, we asked a group of eight novice users and one expert user to measure the distance of the upper two corners of an outdoor window with all different methods. All users were introduced to the system, and all measurements with all methods were repeated three times.

Even for novice users with a short introduction to the system, the results for the support line and support plane method clearly outperform the standard method and the support point method, as indicated by the results listed at the bottom of Table 5.5.

The results in terms of the accuracy of the individual methods for the distance and measurements are depicted in Fig. 5.14. The line and the support plane method consistently and considerably outperform the standard and support point method, and, more importantly, all measurements have a considerably smaller variation.

The users were asked to complete a short questionnaire about the overall usability and the

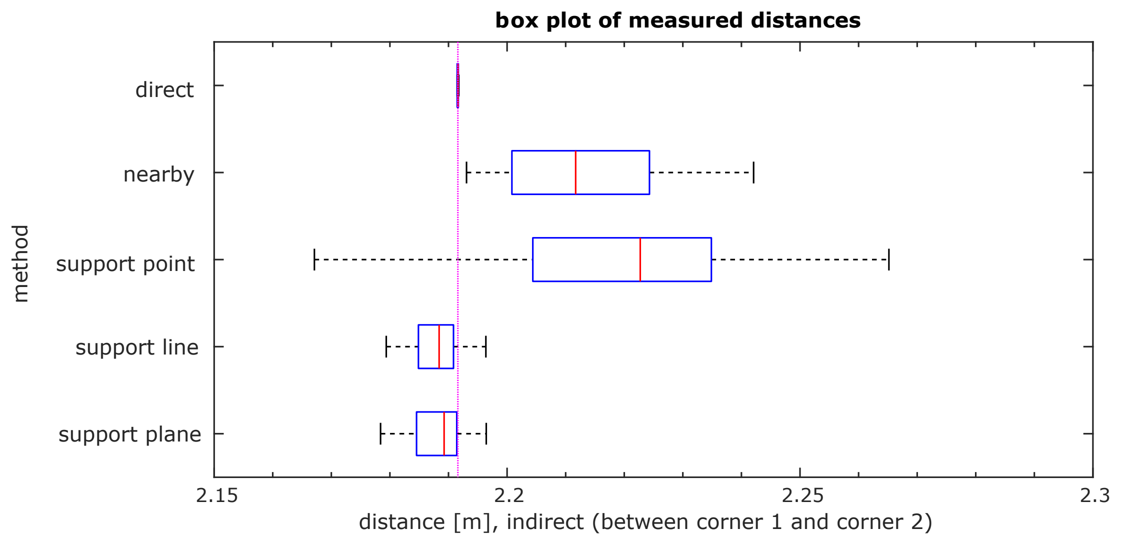


Figure 5.14: Accuracy results [65] for distance measurements between two window corners. The reference distance (horizontal line) was estimated from repeated, perpendicular measurements, using the direct method and modeling clay as temporary planar surface.

intuitiveness of the GUI and the overall approaches. The questions and the answers given by the users are summed in Table 5.6. At a glance, users mainly voted for the support line and support plane method to be favorable over the standard and the support point method in terms of ease of use. Being asked about the usefulness of the three methods introduced in this work, users tended to favor the support line and the support plane method over the support line method. Concerning the accuracy and rapidness of the measurements, users preferred the plane support and the line support method, respectively.

#### 5.2.2.6 Outdoor Measurements

We conducted four outdoor experiments, where we measured the extents of a window from a perpendicular and a $0.25\pi$ point of view. We measured the ground truth distance shown in Figure 5.12 d), f) and h). Then, we applied the four measurement methods again, as discussed in the previous paragraph above.

#### 5.2.2.7 Results

Table 5.5 shows the results of the physical experiments using the indirect evaluation method as discussed in Sec. 5.2.2.1. The Box-and-Whisker plots for some repeated distance measurements are presented in Fig. 5.14. Overall, the support line and support plane based methods achieve considerably better results than the standard method and the support point method, or perform at least on par. The results for an angle of approximately $0.25\pi$ indicate that the support line and support plane based methods achieve considerably better results than the standard method and the support point method for non-orthogonal measurement setups.

## 5.3 Discussion

In this chapter, we described different methods for indirect measurements with an RTS. The results of the simulations, the experiments and the pilot study show that our methods consistently outperform the standard method, even when applied by novice users. One reason for the huge gain in accuracy is due to the definition of the reference method, as the requirement that the projected laser beam has to be fully on the visible surface causes the big systematic error of the measurement method. This is also the main cause for the big systematic error of the reference measurement method.

In the following, we briefly discuss the results for the individual experiments in more detail and draw relationships between the results of the simulation and the physical measurement results with respect to different aspects.

**Planar Target Surface** The MCS results shown in Table 5.4 and Fig. 5.10 indicate the benefits of the proposed indirect measuring methods. For the perpendicular setup, the accuracies of the point, line and plane support methods are comparable with the reference result. We use the direct measurement method with the proposed temporary target modification. For the MCS, the proposed collider extension fulfills the same functionality and allows for direct measurement of edges and corners. All support methods significantly outperform the nearby method, for which we assumed a laser radius of $2.5e-3m$ near the target. In case of an incident angle of $0.25\pi$, the support point method shows a significant systematic error, the support line and plane methods do not suffer from the same error and outperform the other methods. The results from the physical measurements shown in Table 5.5 and in Fig. 5.14 supports our findings.

**Uneven Target Surface** The limitations of the proposed methods are clearly visible when measuring uneven and fillet targets. In this chapter, we used over-determined line and plane fitting, but without outlier-robust estimation. We do not limit the support measurements to the proximity of the IP, but allow the measurements within an ROI with $0.5m$ radius. If we assume no EDM measurement uncertainty, the accuracy of the support methods and the nearby method are in the same range. However, the support line and support plane method show stronger dependencies of the surface properties than the other methods.

**Fillet Target Surfaces** Similar to uneven target surfaces, the support line and plane method are significantly influenced by the surface properties, while the other methods are less affected. Special care must be taken when choosing the best suitable method for a particular measurement. Different ROI radii would provide further insight, but are beyond the scope of this work.

**Nearby and Support Point Method** By definition, both, the nearby and the support point method use a measurement close to the IP. The support point method is designed to reduce the systematic error by applying the angles of the IP while assuming reasonable surface properties. The support point method outperforms the nearby method in all experiments, as shown in Fig. 5.10. This method does not increase the distance measurement count, hence has no significant influence on the measurement duration. Given the low complexity of the algorithm and the user interface, we think the integration into new and existing RTS is reasonable.

**Prototyping and Simulation Environment** The proposed prototyping and simulation environment lowered the implementation effort significantly. Varying physical properties of a measurement setup was easy. The laboratory measurements and our own findings during the physical measurements support this simulation setup for similar hardware configurations. They encourage further work on integrating more realistic sensor models and additional physical properties into the simulator.

**Ray Casting in Simulation** While we used ray casting with a single ray to model the EDM in Unity3D in this chapter, a more realistic simulation would integrate multiple rays, which are distributed within the laser beam. As a side effect, ray casting problems with complex mesh colliders due to round-off errors could be detected and corrected automatically, without the need of the workaround proposed previously.

**Targeting Uncertainty** The proposed UI supports optical a digital zoom for all measurement methods. By zooming in, the targeting uncertainty can be reduced, but it is limited by physical properties of the camera and the measurement setup. An interesting aspect to investigate in the future is the influence of the physical condition operators on the results, such as concentration, distraction, exhaustion or eye strain. In particular, these properties can be modeled as targeting uncertainty, and can be simulated by angle sensor uncertainty.

**Further Work** We identified three major important avenues for future investigation: (a) Outlier detection using of RANSAC schemes. (b) The use of multiple ray casting operations in simulation. (c) The investigation of operator condition effects on measurement errors. (d) Visual feedback and uncertainty indicators for measurements. The former two are targeted more towards improvements of our methods in terms of mathematics and engineering. However, the latter two clearly falls into the HCI domain and are relevant for designing and implementing UI.

We want to emphasize that, despite the basic algorithmic concepts are known for years, practical applications are still largely missing due to the issues arising in real measurement situations. As shown in this chapter, it is therefore highly relevant to study these concepts in practice to identify and overcome shortcomings of the underlying algorithms.

Table 5.5: Distance measurement results of the experiments [65]. For the indirect evaluation method, the error of the average distance between two measured target points is shown. The nearby method, marked in red, is the traditional approach to which we compare.

| $\alpha$ | record | method | $\overline{d}$ $[m]$ | $\hat{\sigma_d}$ $[m]$ | N | $\overline{d_{ref}}$ $[m]$ | $\Delta\overline{d}$ $[m]$ |
|---|---|---|---|---|---|---|---|
| $0.5\pi$ | lab. | direct | 600.191e-3 | 82.942e-6 | 4.000 | 600.191e-3 | 0 |
| | | nearby | 586.664e-3 | 273.151e-6 | 4.000 | 600.191e-3 | 13.527e-3 |
| | | support point | 599.712e-3 | 39.655e-6 | 3.000 | 600.191e-3 | 478.897e-6 |
| | | support line | 599.803e-3 | 866.189e-6 | 5.000 | 600.191e-3 | **387.538e-6** |
| | | support plane | 604.457e-3 | 3.636e-3 | 5.000 | 600.191e-3 | 4.266e-3 |
| | indoor | direct | 881.992e-3 | 362.719e-6 | 10.000 | 881.992e-3 | 0 |
| | | nearby | 893.240e-3 | 820.525e-6 | 8.000 | 881.992e-3 | 11.248e-3 |
| | | support point | 886.912e-3 | 1.921e-3 | 10.000 | 881.992e-3 | 4.920e-3 |
| | | support line | 887.088e-3 | 830.455e-6 | 10.000 | 881.992e-3 | 5.096e-3 |
| | | support plane | 885.561e-3 | 957.555e-6 | 9.000 | 881.992e-3 | **3.569e-3** |
| | outdoor (short) | direct | 883.245e-3 | 25.067e-6 | 4.000 | 883.245e-3 | 0 |
| | | nearby | 888.800e-3 | 14.479e-6 | 3.000 | 883.245e-3 | 5.555e-3 |
| | | support point | 882.519e-3 | 807.959e-6 | 4.000 | 883.245e-3 | 726.362e-6 |
| | | support line | 881.964e-3 | 813.967e-6 | 5.000 | 883.245e-3 | 1.282e-3 |
| | | support plane | 882.181e-3 | 249.838e-6 | 4.000 | 883.245e-3 | **1.065e-3** |
| | outdoor (long) | direct | 2.192 | 107.789e-6 | 10.000 | 2.192 | 0 |
| | | nearby | 2.196 | 1.182e-3 | 10.000 | 2.192 | 4.463e-3 |
| | | support point | 2.193 | 1.248e-3 | 10.000 | 2.192 | 1.761e-3 |
| | | support line | 2.189 | 819.832e-6 | 10.000 | 2.192 | 2.303e-3 |
| | | support plane | 2.190 | 1.212e-3 | 10.000 | 2.192 | **1.587e-3** |
| $0.25\pi$ | lab. | direct | 600.191e-3 | 82.942e-6 | 4.000 | 600.191e-3 | 0 |
| | | nearby | 582.446e-3 | 1.192e-3 | 5.000 | 600.191e-3 | 17.745e-3 |
| | | support point | 584.189e-3 | 240.581e-6 | 4.000 | 600.191e-3 | 16.002e-3 |
| | | support line | 598.194e-3 | 229.861e-6 | 3.000 | 600.191e-3 | 1.997e-3 |
| | | support plane | 598.545e-3 | 654.487e-6 | 5.000 | 600.191e-3 | **1.646e-3** |
| | indoor | direct | 881.702e-3 | 221.990e-6 | 5.000 | 881.702e-3 | 0 |
| | | nearby | 897.636e-3 | 3.285e-3 | 5.000 | 881.702e-3 | 15.934e-3 |
| | | support point | 894.017e-3 | 2.142e-3 | 5.000 | 881.702e-3 | 12.314e-3 |
| | | support line | 882.071e-3 | 607.033e-6 | 5.000 | 881.702e-3 | **369.144e-6** |
| | | support plane | 882.079e-3 | 1.165e-3 | 5.000 | 881.702e-3 | 377.119e-6 |
| | pilot study | direct | 2.192 | 107.789e-6 | 10.000 | 2.192 | 0 |
| | | nearby | 2.214 | 14.497e-3 | 53.000 | 2.192 | 22.456e-3 |
| | | support point | 2.218 | 21.181e-3 | 53.000 | 2.192 | 26.853e-3 |
| | | support line | 2.188 | 4.148e-3 | 54.000 | 2.192 | 3.529e-3 |
| | | support plane | 2.188 | 4.740e-3 | 54.000 | 2.192 | **3.406e-3** |

Table 5.6: Survey results for eight novice and one expert user concerning the ease, usefulness, accuracy and rapidness of the individual methods [65].

| question \ answer | very easy [%] | OK [%] | difficult [%] |
|---|---|---|---|
| How easy was it to use the NEARBY method? | **55.6** | 22.2 | 22.2 |
| How easy was it to use the POINT support method? | **88.9** | 11.1 | 0 |
| How easy was it to use the LINE support method? | **100** | 0 | 0 |
| How easy was it to use the PLANE support method? | **100** | 0 | 0 |

| question \ answer | yes [%] | not sure [%] | no [%] |
|---|---|---|---|
| Do you think the POINT support method is useful? | **44.4** | **44.4** | 11.1 |
| Do you think the LINE support method is useful? | **77.8** | 22.2 | 0 |
| Do you think the PLANE support method is useful? | **88.9** | 11.1 | 0 |

| question \ answer | NEARBY [%] | POINT [%] | LINE [%] | PLANE [%] |
|---|---|---|---|---|
| Which method do you prefer for ACCURATE measurements? | 0 | 0 | 44.4 | **55.6** |
| Which method do you prefer for FAST measurements? | 11.1 | 22.2 | **44.4** | 22.2 |

# 6 Application: RTS-CAD Registration

The accurate registration of an RTS with respect to a given CAD model is a crucial task in the construction industry. One must establish correspondences between control points in the CAD model and measured points in the field. Common registration techniques rely on a reference network of control points in the CAD model. Usually physical markers or natural points of interest are selected as control points. Physical markers can be reflective targets with known measurement properties or non-reflective targets like geodetic marks or cast metal disks.

Especially when planning or documenting installations, planning reconstructions or placing physical markers for assembly drills, fast RTS registration without dedicated control points is desired. Common natural measurement targets are building corners, edges or steeples. They have a high recall value and avoid physical installation of targets. However, the accuracy and reliability of such natural targets are influenced by the distance measurement problems of non-planar targets as described in the previous chapter.

In this chapter, we present a user-guided algorithm for simple and efficient registration of an RTS with a polygonal CAD model in indoor environments without the need for control points. The user interaction is reduced to selecting a local Manhattan-like corner structure for initial model alignment; accurate registration of the device is carried out automatically. Our algorithm relies on angle and distance measurements only and, therefore, is not limited to vision-based RTS. The registration is performed with a sparse 3D point cloud scan to keep the measurement duration to an acceptable level. Furthermore, the critical distance measurement of non-planar targets can be avoided to decrease the registration uncertainty.

In particular, we propose a robust Manhattan corner extraction for ICP initialization, a robust sample selection method for ICP refinement, and a sorting algorithm to optimize the scan order of the measurements. With our algorithm, it is possible to reduce the user interaction significantly, while retaining an accurate registration without the usage of reflective targets. The proposed approach simplifies the registration procedure and relaxes

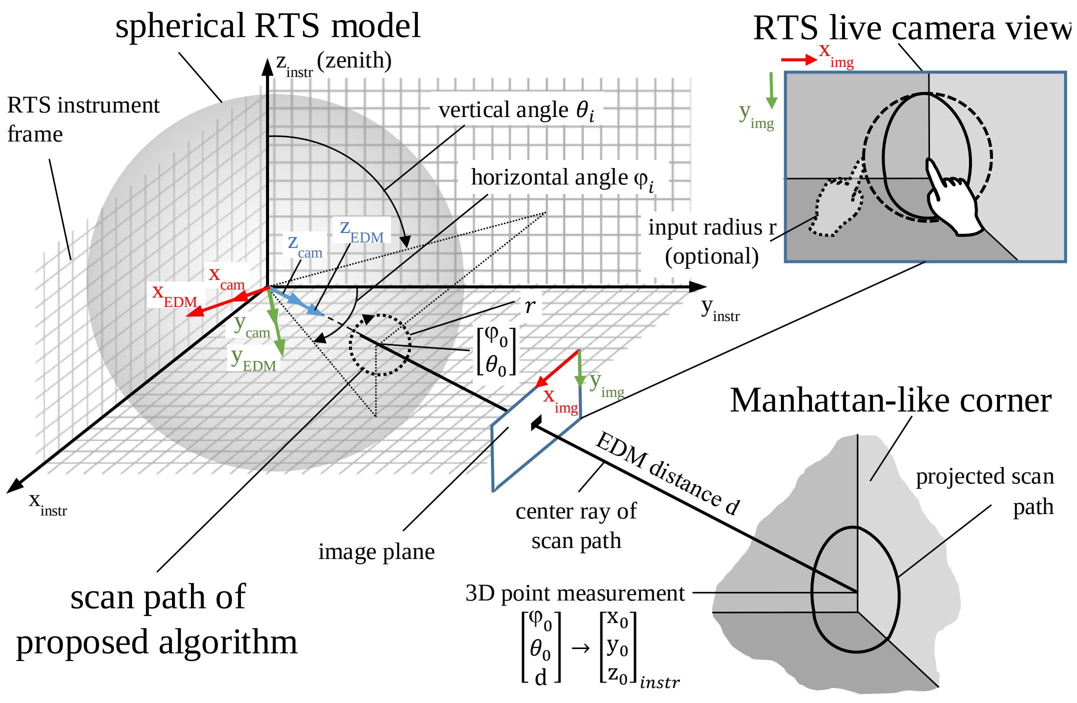


Figure 6.1: Idealized geometric model of a calibrated RTS [64]. The spherical coordinate frame of the RTS measurements, the EDM frame and the camera coordinate frame are aligned in this particular model.

the accuracy constraints for initial measurements. This is crucial for non-experts to perform reliable and robust measurements. We avoid computationally expensive calculations to enable a real-time implementation on embedded systems and mobile devices.

This chapter is based the work by Klug et al. [64], but provides a more complete description.

## 6.1 RTS Registration

The registration of the current position of the RTS with respect to a CAD model of the environment can be reduced to a generic 3D registration problem. Traditional 3D registration algorithms use three or more point correspondences to estimate the pose between two models [47]. The ICP algorithm and it variants are widely used for the object registration problem. However, carefully selected sample points and a good initialization pose are required, especially when dealing with sparse point clouds.

Our algorithm can determine the registration without dedicated control points by using local geometry for initial pose estimation and an ICP based sampling and refinement step.

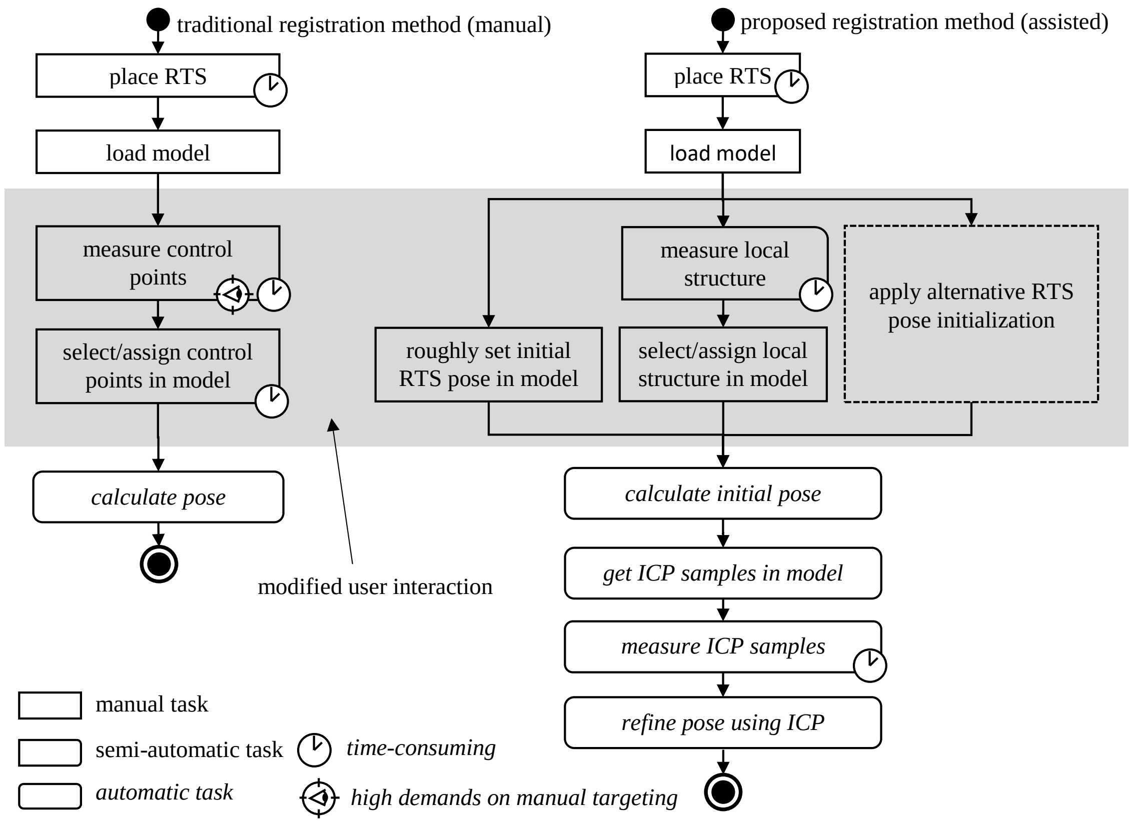

traditional registration method (manual)
place RTS
load model
measure control points
select/assign control points in model
calculate pose
proposed registration method (assisted)
place RTS
load model
measure local structure
roughly set initial RTS pose in model
select/assign local structure in model
apply alternative RTS pose initialization
calculate initial pose
get ICP samples in model
measure ICP samples
refine pose using ICP
modified user interaction
manual task
semi-automatic task
automatic task
time-consuming
high demands on manual targeting

Figure 6.2: Manual registration workflow (left) compared to workflow of proposed CAD based registration (right). The manual workflow requires visible and CAD-registered control points, whereas control points are optional for the proposed method.

Alternatively, the initial pose can be roughly defined by the operator using the CAD model.

Figure 6.2 compares the manual registration workflow with our assisted approach, which uses local geometry instead of control points. Manual registration requires to create and select control points in the model with a CAD software, to target and measure them physically with an RTS, and to define point correspondences between the selection and the measurement. In contrast, when using the assisted method, the user positions the RTS, loads a CAD model and roughly specifies the initial pose. Then, by triggering the registration algorithm, measurement samples are taken automatically to refine the initial registration. Note that the initial registration is also referred as *coarse registration* throughout this work.

### 6.1.1 Manual RTS Registration

When registering an RTS with respect to a CAD model manually, the user is responsible for measuring and assigning the control points. The steps for this approach include: a) place the total station in the environment, b) load the model, c) create control points in the model, d) measure the control points, e) assign the measured control points to the reference measurements, and f) calculate the current pose of the total station with respect to the model frame. In the simplest case, this problem can be mathematically formalized as point cloud fitting problem with known point correspondences, using a six DOF Euclidean transformation.

Let $\mathbf{x}'_i$ be a CAD point, $\mathbf{x}_i$ be the corresponding measured 3D point, and $(\mathbf{R}, \mathbf{t})$ be the relative Euclidean transformation between the measurement frame and the CAD frame. Then, the relative transformation between the two frames is given by

$$\mathbf{x}'_i = \mathbf{R} \cdot \mathbf{x}_i + \mathbf{t} \qquad (\mathbf{R}, \mathbf{t}) = \arg\min_{\mathbf{R},\mathbf{t}} \sum_{i=1}^{m} ||\mathbf{R} \cdot \mathbf{x}_i + \mathbf{t} - \mathbf{x}'_i||^2 \tag{6.1}$$

with the point correspondence count $m \geq 3$. The problem is known as *best rigid body transformation in least square sense* with a well-established closed-form solution proposed by Horn [46]. In practice, outliers can be handled using the RANSAC framework proposed by Fischer *et al.* [37].

### 6.1.2 Assisted RTS Registration

Assisted RTS registration with respect to a CAD model requires initial values for the rotation $\mathbf{R}$ and the translation $\mathbf{t}$. Our method supports the user in defining an approximate initialization pose and then refines the result automatically. A good initial registration pose is required for fast and robust registration. A simple, yet robust method for pose initialization is defining the initial pose manually in the CAD model.

Another pose initialization method is measuring a local corner structure and matching its pose with respect to a selected CAD region. An RTS provides accurate single point measurements with low sampling rate. Therefore, we explicitly avoid dense point cloud scans to keep the registration reasonable fast. While the ICP is an established solution for the registration of point clouds and meshes, the sparsity of the measurements is critical for ICP based pose estimation without providing further user input. The sparsity of the point cloud is mainly driven by keeping the sampling time acceptable. In this work, we experienced an EDM update rate of about 0.5...20 Hz for repeated measurement of the same target, but lower update rates for different targets. This is consistent with the specified distance sampling rates of common RTS shown in Table 1.1.

We experimentally determined an acceptable sample count for the course registration of $N_s \leq 40$ points in indoor environments. Multiple initial pose candidates would be required for a robust pose estimation, which comes at the expense of additional computation time.

The method proposed by Nguyen et al. [79] uses sparse 3D point clouds, graph based segmentation and EM for robust plane detection. Our algorithm is based on the work of Nguyen et al. However, with our setup, certain modifications are required: 1. The mobile EDM is replaced by an RTS. Hence, no SLAM system is required for connecting

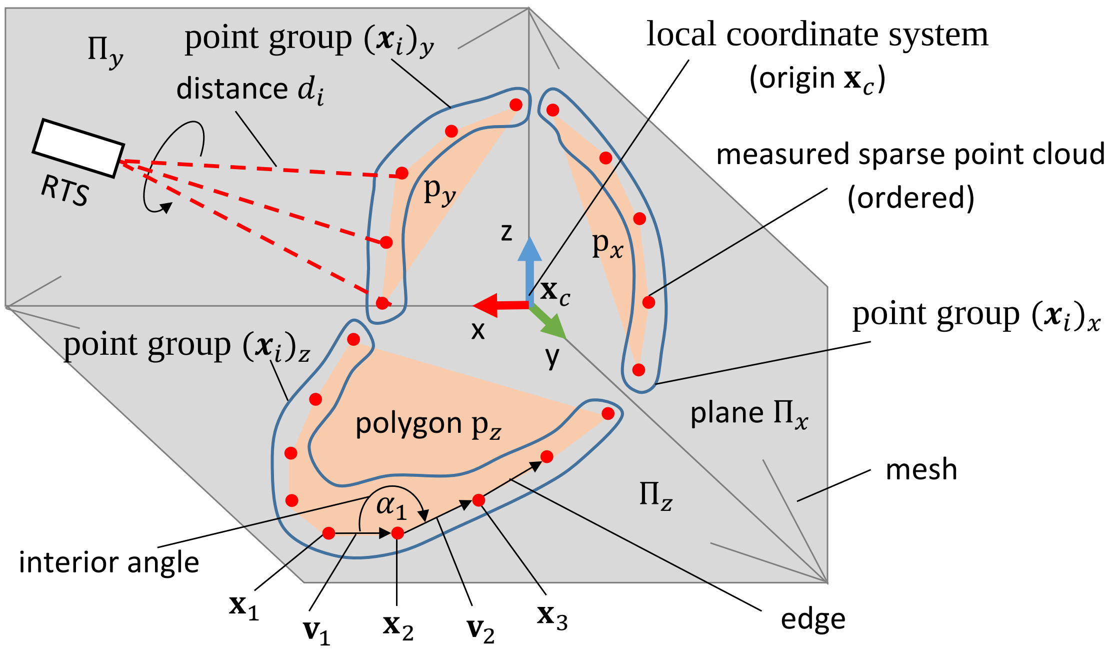


Figure 6.3: Manhattan-like structure detection. Once detected, the initial RTS registration with respect to the CAD model is done by aligning a local coordinate system of the measured and selected structure. The local coordinate axes are automatically extracted from both, the CAD model and the measured point cloud.

multiple measurements. 2. 3D points can be measured with higher accuracy and with a predefined scan order. This reduces the optimization complexity and allows for a more robust algorithm. 3. Realistic indoor scenarios and usage on construction sites demand high stability with low contrast images. Hence, the point segmentation proposed by the authors cannot be used. 4. Pose estimation of a mesh with respect to a point cloud was not applied by the algorithm and must be addressed separately.

In the following, we provide modifications to the method proposed by Nguyen et al., which lead to a simple, yet robust method for Manhattan-like corner pose estimation for RTS in low contrast scenarios. The RTS camera is only used to steer the RTS to the area of interest, but not for segmentation. The camera can be removed completely for setups with different hardware control interfaces.

### 6.1.3 Local Manhattan-Like Corner Estimation With Ordered Sparse Point Clouds

First, the proposed algorithm defines a measurement path for the RTS for creating a ordered set 3D point measurements. Using the measured point set, plane detection is applied to classify the sparse scan of a local Manhattan-like structure into following three categories: single wall, 3D edge with two visible planes, or 3D corner with three visible

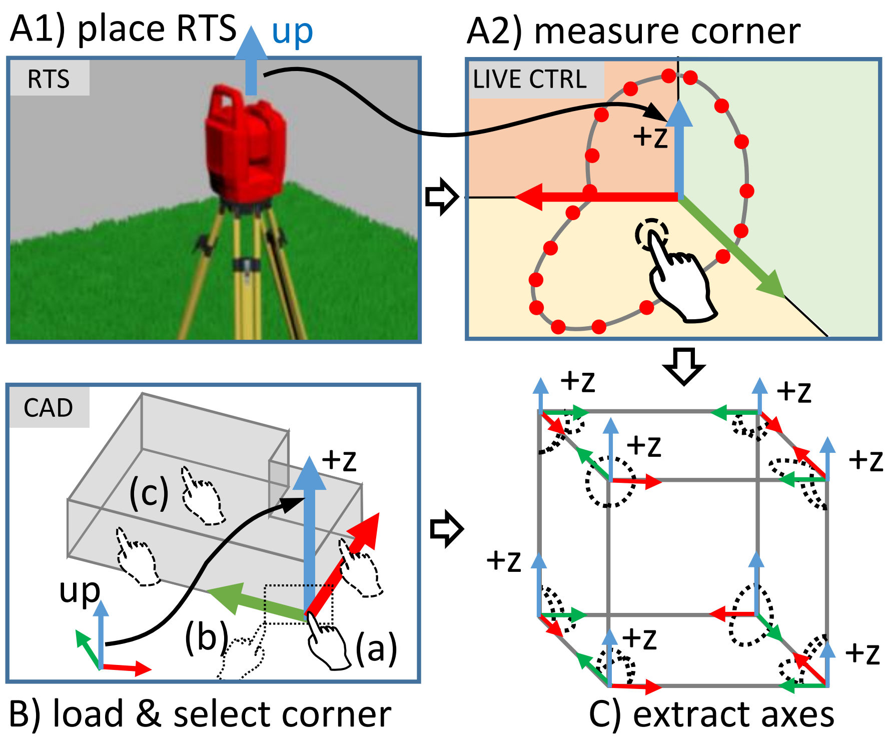


Figure 6.4: Local coordinate axes extraction and assignment. The up-vector of the scene and the up-vector of the RTS are used to resolve the ambiguity of the direction and label of the z-axis. Selection methods: (a) select corner by vertex; (b) select corner area; (c) select three surfaces. The task flow A1-A2-C can be executed parallel to the task flow B-C to reduce the idle time of the user.

planes. In the current work, only the latter is accepted as valid measurement. The complete plane extraction algorithm consists of following steps: 1. Automatically measure the local area of a target using a circular movement, 2. create initial point set groups 3. refine groups using EM, and 4. extract planes using the RANSAC framework for each point group. Figure 6.1 shows the circular scan path, the projected scan path at the measured corner, and an exemplary RTS control UI for the algorithm.

Discrete sample points form a sparse point cloud around the current target. The points are recorded by an RTS using a circular EDM motion. The RTS angle control parameters $\begin{bmatrix}\varphi_i & \theta_i\end{bmatrix}^T$ are defined by

$$\begin{bmatrix}\varphi_i\\ \theta_i\end{bmatrix} = \begin{bmatrix}\varphi_0\\ \theta_0\end{bmatrix} + r \cdot \begin{bmatrix}cos(\omega_i)\\ -sin(\omega_i)\end{bmatrix} \qquad i = 1...N_s,\ \omega_i = i\frac{2\pi}{N_s+1} \tag{6.2}$$

where $\varphi_0$ and $\theta_0$ are the horizontal and vertical angle of the center of the measurement, $\varphi_i$ and $\theta_i$ are the spherical coordinates of sample point $\mathbf{x}_i$, and $N_s$ is the sample count.

**Algorithm 1:** Peak detection algorithm with non-maximum suppression of successive points. The non-maximum suppression in this algorithm is limited to a single subsequent edge. The angle threshold is set to $\alpha_{thr} = \frac{\pi}{4}$ (experimentally determined).

```
function peakfind
Input  : $angles := (\tilde{\alpha}_1, \dots, \tilde{\alpha}_{Ns})$
Output: $peak\_indices$
state s := $NO\_PLATEAU$
foreach $i$ in $(1, 2, \dots, N_s)$ do
    if $s = NO\_PLATEAU$ AND $\tilde{\alpha}_i > \alpha_{thr}$ then
        APPEND $i$ to $peak\_indices$
        s := $PLATEAU$
    else
        s := $NO\_PLATEAU$
return $peak\_indices$
```

**Algorithm 2:** Simple greedy non-maximum suppression algorithm based on the size of the point groups. List boundary handling is omitted for reasons of readability.

```
function nonmaximumsup
Input  : $peak\_indices$
Output: $filtered\_peak\_indices$
state s = $NO\_PLATEAU$
var $idx_{prev}$ = 0
foreach $idx$ in $peak\_indices$ do
    if $(idx - idx_{prev}) > 3$ then
        APPEND $idx$ to $filtered\_peak\_indices$
        $idx_{prev} = idx$
return $filtered\_peak\_indices$
```

By steering the RTS approximately to the corner, the measurement center angles $\varphi_0$ and $\theta_0$ are implicitly defined by the current RTS pose. The radius $r$ defines the swing of the spherical RTS control parameters for the local measurement. When defined visually, the angle between the center ray and the back-projected ray of a user defined image point is used. However, $r$ can also be derived from a predefined spherical bounding box at the corner, using an initial measured EDM distance, which further reduces the required user input.

The proposed classification accepts simple Manhattan-like structures with one, two or three main planar areas.

The point cloud, consisting of $N_s$ measured points $\mathbf{x}_i$, is interpreted as closed polygon $K = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{Ns}, \mathbf{x}_1)$ for pre-processing and initial point grouping. Besides image-based segmentation, Nguyen et al. use EDM distances for initial point grouping. In contrast, RTS provide not only distances, but 3D point measurements, which we use for segmentation.

Let the interior angle $\alpha_i$ be the angle between two successive measured points, interpreted as 3D polygon vertices. Then, $\alpha_i$ is given by

$$\alpha_i = \arccos(\mathbf{v}_i \cdot \mathbf{v}_{i-1}) \qquad \mathbf{v}_i = \frac{\mathbf{x}_{i+1} - \mathbf{x}_i}{||\mathbf{x}_{i+1} - \mathbf{x}_i||} \tag{6.3}$$

where $\mathbf{v}_i$ is the direction from vertex $\mathbf{x}_i$ to vertex $\mathbf{x}_{i+1}$. To identify initial coplanar point groups, we subtract a systematic angle $\alpha_s$ according to

$$\tilde{\alpha}_i = \max(|\alpha_i| - \alpha_s, 0) \qquad \alpha_s = \frac{2 \cdot \pi}{N_s + 1} \tag{6.4}$$

The systematic angle $\alpha_s$ describes the angle between two consecutive edges for a perpendicularly measured, planar target. We perform peak detection of the absolute values of the remaining angles with downstream non-maximum suppression of the detected plateaus. The peak detection algorithm is shown in Alg. 1. After detection, groups with less than three points are joined using the greedy iterative algorithm shown in Alg. 2. EM optimization is applied for robust plane detection. In the Expectation step, measurements are assigned to coplanar point groups. However, the sorted input allows for k-means clustering in the Expectation step rather than using Gaussian Mixture Models as proposed by Nguyen et al. [79]. Let the plane $\Pi_j$ be the estimated geometric model of point group $j$. The initial likelihood $p$ of a homogeneous measurement point $\tilde{\mathbf{x}}_i$ to belong to the plane $\Pi_j$ is obtained from Alg. 2. In the subsequent iterations, the likelihood is updated using

$$p(\tilde{\mathbf{x}}_i \in \Pi_j | \Pi_j) = \begin{cases} 1, & j = \arg\min_k(\Pi_k^T \cdot \tilde{\mathbf{x}}_i),\ |j - m| \leq 1 \\ 0, & \text{otherwise} \end{cases} \tag{6.5}$$

where $\Pi_k$ is a $4 \times 1$ vector, which contains the plane parameters for the co-planar point group $k$. In the previous iteration, point $\mathbf{x}_i$ was assigned to plane $\Pi_m$. In the Maximization step, plane fitting and merging of similar planes is applied. However, in all algorithm steps, only connected groups are considered. This reduces the effects of invalid inter-class point assignments and invalid plane merging. The benefits of the ordered point clouds can be compared to the benefits of *organized point clouds* over *unorganized point clouds*, as discussed by Trevor et al. [104]. Figure 6.3 shows an exemplary measurement. Different representations of the Manhatten-like structure are used for the individual algorithm steps. Hence, the local structure is represented as single mesh, or as connected planes, polygons, or grouped points.

From the proposed plane detection, three planes are selected and classified as a corner structure. First, the most prominent plane $\Pi_i$ is selected, using the inlier count $N(i)$ of the plane estimation. Then, the two remaining planes $\Pi_j$ and $\Pi_k$ are selected by maximizing the span and inlier count of the three planes. Hence, the optimization problem is given by:

$$\left.\begin{array}{ll}
i = \underset{i}{\operatorname{argmax}}(N(i)), & (j,k) = \underset{j,k}{\operatorname{argmax}}(C_c), \text{ subject to:} \\
C_c = c_1 C_1 + c_2 C_2 & 0 \le (c_1, c_2) \le 1 \\
C_1 = (\mathbf{n}_i \times \mathbf{n}_j) \cdot \mathbf{n}_k & C_2 = \dfrac{N(i) + N(j) + N(k)}{N_s} \\
i \neq j \neq k & 3 \le (N(i), N(j), N(k)) \le N_s
\end{array}\right\} \tag{6.6}$$

where $N(l)$ is the inlier count for some plane $l$; $C_c$ is the fitness function, defined as weighted sum of $C_1$ and $C_2$; $C_1$ maximizes the span, and $C_2$ maximizes the inlier count of the planes $\Pi_i$, $\Pi_j$ and $\Pi_k$. The weights $c_1$ and $c_2$ were experimentally determined and were set to $c_1 = c_2 = 0.5$.

For subsequent alignment, a right-handed local coordinate system as shown in Figure 6.4 is defined, which is called *local corner frame* in the following. The origin $\mathbf{x}_c$ is given by the intersection of the planes, the axes are given by orthonormalizing the plane normals $\mathbf{n}_i$, $\mathbf{n}_j$ and $\mathbf{n}_k$. For the economy of notation, the plane normals and the related orthonormal vectors are denoted by the same variables[1]. The initial RTS pose can be estimated by relating the local corner frame of the selected CAD model and the measured corner. At this state, only the orientations of the axes are known, but not the labels and directions. Hence, a pose ambiguity with 24 possibilities must be resolved. The correspondences between axis labels and the extracted plane normals can either be selected manually or assigned automatically. Automatic assignment requires additional constraints; the up-vectors for both the CAD model and the RTS must approximately be known. As RTS require upright operation, and surveying models usually contain the information about the up-vector, no additional user input is required. Without loss of generality, we assume that the up-directions of the CAD model and the RTS are both aligned with the z-axis of the particular coordinate system. In the proposed setup, Manhattan-like corners with three planes usually have a prominent z-plane. Thus, from the previous detected corner planes, the z-plane can be estimated by:

$$i_z = \underset{l}{\operatorname{argmax}} |\arccos(\mathbf{z} \cdot \mathbf{n}_l)| \qquad l \in (i, j, k) \tag{6.7}$$

where $i_z$ is the index of the z-plane, and $\mathbf{n}_l$ is the plane normal of the extracted planes $\Pi_i$, $\Pi_j$ or $\Pi_k$. Eqn. 6.7 can be applied to both the selected CAD corner and the measured corner. This reduces the remaining alignment ambiguity to four possibilities as shown in Figure 6.4.

The measured RTS samples are mainly distributed in one octant of the local corner

[1]While the interpretation of the plane intersection and normals as local coordinate system enforces orthonormalization, the latter is not mandatory for the proposed algorithm.

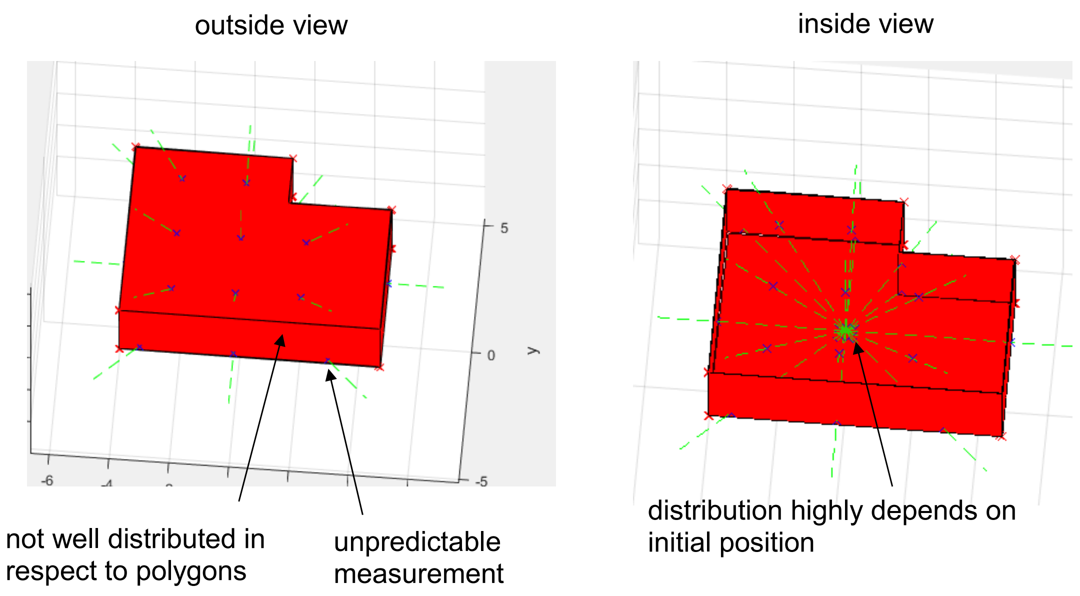

outside view
inside view
5
0
y
-5
-6
-4
not well distributed in respect to polygons
unpredictable measurement
distribution highly depends on initial position

Figure 6.5: Sampling problems. Sample distribution of standard uniform sampling method distribution highly depends on initial registration position.

frame. The remaining ambiguity can be resolved for the measured corner by enforcing that the majority of the measured samples have to be in the positive half-spaces of the local corner frame. Similarly, the alignment of the local corner frame of the selected CAD area can be fixed. Let each vertex of the selection vote for a half-space. In case of ambiguities votes, additional user input is required. Otherwise, the same method as for the measured corner is applied.

Finally, the measured and selected local corner frame can be aligned using Eqn. 6.1. Optionally, ICP can be applied to the selected corner and the measured samples to refine the initial RTS pose.

### 6.1.4 Automatic Pose Refinement Using Additional Samples

The estimated RTS pose of the proposed initialization method relies on local measurements only. Hence, measurement uncertainties and discrepancies between the model and reality may lead to an unacceptable registration error. In the following, iterative pose refinement is discussed, using the previous results as initialization. In general, the pose refinement step of the proposed registration algorithm automatically

1. selects robust and well distributed sampling points for further pose refinement,
2. calculates the minimum movement for the RTS to measure the sampling points,
3. measures the selected samples, and

4. refines the registration pose.

The following sections describe the individual algorithm steps.

### 6.1.5 Robust Sampling

For pose refinement, additional sample points are required. The sample points have to be visible from the current RTS position. At this state, only the RTS pose from the initial registration can be used to pick sample candidates from potential visible regions of the CAD model. Without prior knowledge about the scene, the following sampling strategies could be applied: a) random or uniform sampling within the bounding box of the CAD model, b) random or uniform sampling on a unit sphere around the initial RTS pose, c) random or uniform sampling of points of the surface of the CAD model. Random sampling without guidance would lead to an unacceptable measurement time, even with a modern RTS. Furthermore, without incorporating the initial pose, the spatial distribution of random or uniform distributed samples highly depends on the scene setup, as shown in Figure 6.5. In this section, we provide a better sampling strategy by incorporating the CAD model and the initial RTS pose. First, we create a potentially visibility set of polygons. This step extracts polygons of the CAD model that are visible from the initial RTS pose. Connected, co-planar polygons are joined together for further sampling.

Then, we define sample candidates as uniformly distributed point sets for each polygon. Indoor environments usually have joints, skirting boards and similar details, which are not present in CAD models. Juretzko [60] describes problems with edge and corner measurements in reflectorless RTS mode. Klug et al. [65] substantiate the statements of Juretzko by analyzing measurement errors of Manhattan-like corners. For these reasons, the pose refinement step excludes sample candidates near polygon edges explicitly. Figure 6.6 shows sampling candidates, which are calculated, filtered and measured by the pose refinement step.

Then, the proposed sampling method uses a greedy point-picking algorithm, which iterates over polygons to select well distributed samples. The procedure terminates after a predefined number of samples. Alg. 3 shows the pseudo-code for picking sample points.

Finally, the extracted sample coordinates are converted to spherical coordinate angles for controlling the RTS.

### 6.1.6 Efficient Sample Measurement Order

Measuring random samples with the RTS without ordering would lead to undesired and unnecessary RTS movements. Proper definition of the scan order decreases the power consumption of the device, increases laser safety on construction side, and lowers the overall measurement time. Furthermore, it increases the aesthetic aspect of the RTS operation by using smooth movements rather than edgy jumps during the registration.

Finding the minimum distance between multiple stops is known as the traveling salesman problem (TSP) One common formalization is based on linear integer programming

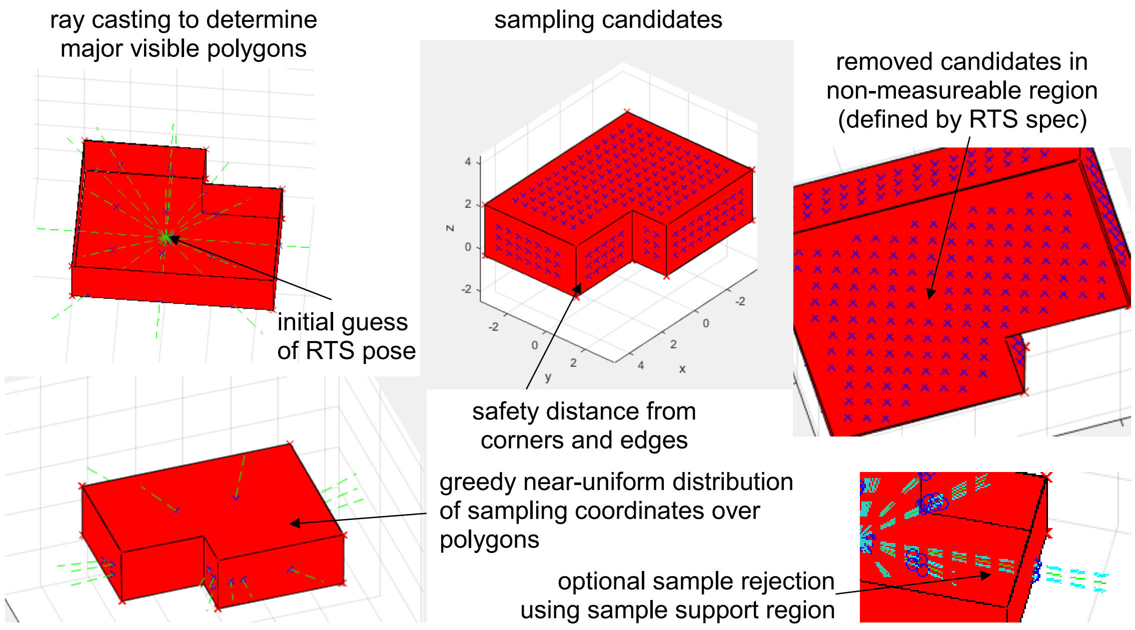

ray casting to determine major visible polygons
initial guess of RTS pose
sampling candidates
z
x
y
safety distance from corners and edges
removed candidates in non-measureable region (defined by RTS spec)
greedy near-uniform distribution of sampling coordinates over polygons
optional sample rejection using sample support region

Figure 6.6: Robust sample extraction using initial pose and visible CAD polygons that are used to control the RTS.

(ILP)

$$x_{ij} = \begin{cases} 1 \text{ connection between point i and j} \\ 0 \text{ otherwise} \end{cases} \tag{6.8}$$

where $x_{ij}$ describes the movement from point $i$ to point $j$. The shortest distance between two points on a sphere can be expressed in spherical coordinates by the *great-circle distance*

$$\Delta\alpha = \arccos(\sin\theta_i \sin\theta_j + \cos\theta_i \cos\theta_j \cos(\varphi_i - \varphi_j)) \qquad d_{ij} = r \cdot \Delta\alpha \qquad r = 1 \tag{6.9}$$

where $\Delta\alpha$ is the central angle between points $i$ and $j$, $d_{ij}$ is the shortest distance between them, and $r$ is the radius of the sphere.

The ILP for $N_s$ measurement points can be written as [107]

$$\left.\begin{array}{l} \min_{x_{ij}} \sum_1^{N_s} \sum_1^{N_s} d_{ij} x_{ij}, \text{ subject to:} \\ x_{ij} \in \{0,1\} \qquad 1 \le \{i,j\} \le N_s \\ \sum_{i=1,i\neq j}^{N_s} x_{ij} = 1 \qquad \sum_{j=1,j\neq i}^{N_s} x_{ij} = 1 \end{array}\right\} \tag{6.10}$$

which leads to a smooth sampling path and decreases the overall sampling time. Note

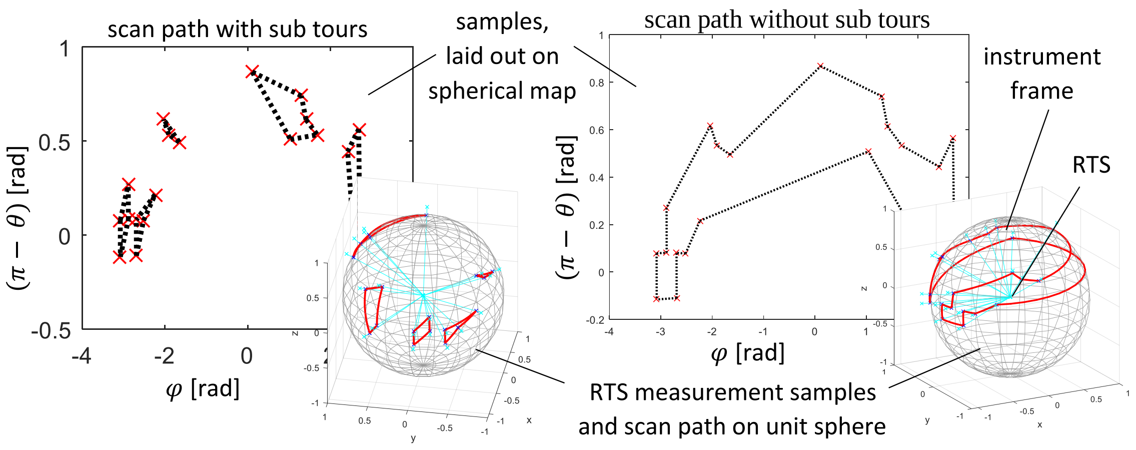


Figure 6.7: Scan path optimization of an exemplary collection of samples, with and without sub-routines.

that sub-tours are explicitly allowed for performance reasons. Figure 6.7 visualizes the optimization of an exemplary collection of points to measure with and without sub-routines.

### 6.1.7 ICP Pose Refinement

The problem of refining the initial RTS pose with respect to the CAD model is a *pairwise surface-based registration* problem. A well-studied solution is the ICP algorithm. In this work, we used the ICP algorithm with measured points, model surfaces, and the point-to-plane distance metric as described by Mehdi [76].

## 6.2 Experiments

The proposed method is mainly designed for usage in indoor scenarios, where Manhattan-like corners are present and measurable. Physical evaluation setups contain different error sources, which contribute to the overall measurement uncertainty for both, the reference results and the algorithms under test. In general, the uncertainty in the result of a measurement or calculation, which consists of multiple components, is known as *combined standard uncertainty*. The estimation, propagation and assessment of the measured physical quantities and the related uncertainties are described in the JCGM 100:2008 GUM [57].

Without controlling the properties of the physical setup, the uncertainty of the result of a measurement or calculation cannot be separated reliably into effects of different error sources. This renders a physical evaluation approach critical for the in-depth analysis of the proposed algorithm.

For this reason, we used a simulated RTS setup for the evaluation in this work. Hence, effects of sensor noise and as-built discrepancies with respect to the registration results could be analyzed.

In the following, we provide the description of the used test data and the test coverage for the proposed algorithm, the simulated RTS measurement setup and the variations of the uncertainty sources for the RTS sensors and the CAD models.

### 6.2.1 Test Setup and Test Coverage

We altered several test parameters for better test coverage as shown in the test taxonomy in Table 6.1. In particular, we compared different registration methods, different levels of as-built discrepancies, and different sensor noise settings. We report all registration results with and without subsequent ICP refinement for better comparison of intermediate results.

The RTS control parameters and the reference result for the registration were automatically generated for each test. The CAD models and their variants were generated semi-automatically. The model variants were used to simulate as-built discrepancies. We compared the reference RTS registration with the manual registration method and with the assisted registration method. For manual registration, we used three point correspondences for each test. We also applied the ICP approach to the manual registration for better comparison. Here, we showed that the corner estimation of the workflow proposed in Figure 6.2 could easily be replaced by any other initialization method, like using three point correspondences.

The individual test sets analyze different as-built discrepancies. We analyzed 1. no as-built discrepancies, 2. as-built discrepancies at corners with 15mm and 30mm curva-

**Algorithm 3:** Greedy point-picking algorithm. Points are selected from each polygon group. Connected co-planar CAD areas are joined before point-picking is applied.

```
function select_sample_points;
Input  : sample count threshold $sample\_cnt$ and sample candidates with reference to
         polygons $p_{cand}$
Output: selected sample points $p_{sel}$
list $p_{sel}$ := () // start with empty list
foreach $poly$ in $polygons$ do
    $p := RANDOM\_SAMPLE(poly, p_{cand})$
    if $p = ()$ then
        // no more candidates for $poly$
        REMOVE $poly$ from $polygons$
    else
        APPEND $p$ to $p_{sel}$
        REMOVE $p$ from $p_{cand}$
        if $COUNT(p_{sel}) = sample\_cnt$ then
            BREAK
return $p_{sel}$
```

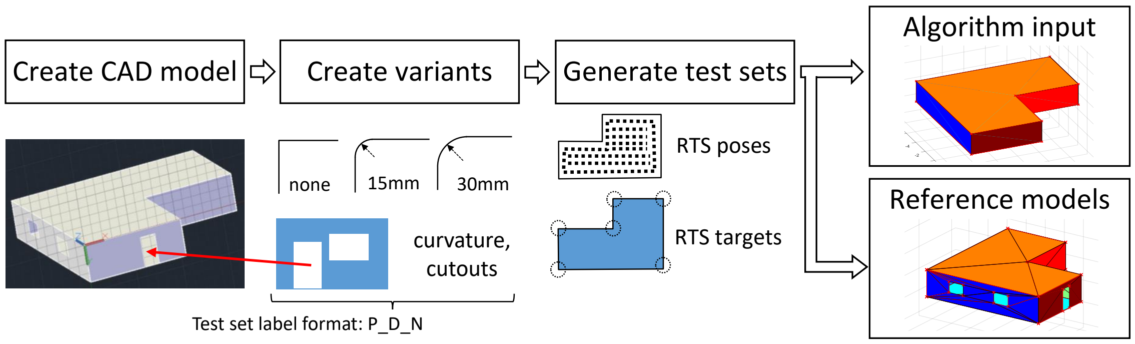


Figure 6.8: Test data generation, using single-source for different CAD model variants. As-built variants include fillets and curvature at edges and missing details like door and window cutouts.

ture radius, 3. as-built discrepancies at faces with missing cutouts. In case of as-built discrepancies, the CAD model did not have the same detail level as the reality. We simulated the scenarios with detailed CAD models, but used the simplified CAD model in our registration algorithm. Furthermore, we simulated measurement uncertainties for sensors and targeting, using the settings given in Table 6.2. The EDM measurement uncertainty $u_c(d)$ and the angle and targeting uncertainty $u_c(\alpha)$ were simulated by

$$u_c(x) \approx \sqrt{u_1(x)^2 + (x u_2(x))^2} \qquad x \in \{d, \alpha\} \tag{6.11}$$

where $u_c(x)$ is the simulated standard uncertainty with normal distribution and zero mean; $u_1(x)$ is the additive and $u_2(x)$ is the proportional uncertainty component. $u_c(\alpha)$ is the angle sensor and targeting uncertainty, which is assumed to be equal for horizontal and vertical angle.

The steps required for test data generation are shown in Figure 6.8. During evaluation, the registration algorithm and the reference simulation can use different variants of the same CAD model for analyzing effects of as-built discrepancies. We used detailed models for simulation and models with as-built discrepancies as input for the discussed algorithms. The simulated as-built discrepancies include missing CAD model details, such as edge curvature, windows and door cutouts.

The error metric used in this work is given as follows: Let the registration error of the RTS with respect to the CAD model be the difference in angle and position between the true and the calculated RTS frame. Then, the position error $e_t$ and the rotation error $e_r$ for test set $j$ are given by

$$e_t = \sqrt{\sum_{i=1}^{N} \frac{||\mathbf{y}_i - \mathbf{y}'_i||^2}{N}} \qquad e_r = \sqrt{\sum_{i=1}^{N} \frac{(\Delta\alpha_i)^2}{N}} \tag{6.12}$$

where $N$ is the test count of the test set $j$; $i$ is the test index within the test set; $\mathbf{y}_i$ and $\mathbf{y}'_i$ are the true and the estimated position of the RTS, respectively; $\Delta\alpha_i$ is the minimum rotation between the true and the estimated RTS pose. The minimum rotation is given by the relative rotation between the calculated and the true RTS frame, using the rotational part of the corresponding axis-angle representation. In addition, we calculated the standard uncertainties $u_c(e_t)$ and $u_c(e_t)$ according to GUM [57]. RTS sensor simulation with Unity3D uses single-precision floating-point arithmetic and causes outliers in the test results due to round-off effects and EDM ray-casting misses as discussed in Sec. 5.2.1.3. Here, we used the median absolute deviation to reduce the influence of outliers [65, 71] rather than modifying the model as described in Fig. 5.9. In particular, we applied the Matlab function *isoutlier* with default parameters. Each test set contains 130 tests. On average, we removed $\approx 11\%$ outliers from each set. Test set labels are defined in Table 6.1. In addition, we defined combined test sets for the as-built discrepancies for better comparison. In particular, we applied Eqn. 6.12 on the collected values for $||\mathbf{y}_i - \mathbf{y}'_i||$ and $\Delta\alpha_i$ of the different as-built discrepancy test sets.

Table 6.1: Test set taxonomy and label encoding; test set label format: $\{P\_D\_N\}$. Example: $c\_d00\_n0$ denotes the test set, which uses the corner estimation method as initialization, no as-built discrepancies and no sensor noise.

| name | variable | value | description |
|---|---|---|---|
| initialization method | P | p | 3 points |
| | | c | corner |
| refinement method | | | none |
| | | | ICP |
| as-built discrepancy | D | d00 | none |
| | | d15 | 15mm curvature |
| | | d30 | 30mm curvature |
| | | dcu | cutouts |
| noise | N | n0 | None |
| | | n1 | measurement uncertainty |

Table 6.2: Sensor noise settings, modeled as measurement uncertainties, enumerated by the test set label format variable $N$.

| label | description | $u_1(d)$ | $u_2(d)$ | $u_1(\alpha)$ | $u_2(\alpha)$ |
|---|---|---|---|---|---|
| n0 | without noise | 0 | 0 | 0 | 0 |
| n1 | with noise | $0.75e-3$m | $10e-6$m | $\frac{5}{3600} \cdot \frac{2\pi}{360}$ rad | 0 |

Table 6.3: Evaluation results, using 130 tests for each test set. Test set labels defined in Table 6.1.

| test set | | initialization | | ICP | |
|---|---|---|---|---|---|
| No. | label | $e_t$ [m] | $e_r$ [rad] | $e_t$ [m] | $e_r$ [rad] |
| 1 | p_n0_d00 | $\mathbf{6.55e-06 \pm 6.14e-11}$ | $\mathbf{9.51e-07 \pm 1.46e-12}$ | $8.10e-07 \pm 5.67e-13$ | $3.01e-07 \pm 5.36e-14$ |
| 2 | c_n0_d00 | $2.29e-05 \pm 7.30e-10$ | $3.69e-06 \pm 1.26e-11$ | $\mathbf{1.02e-06 \pm 7.62e-13}$ | $\mathbf{3.55e-07 \pm 8.05e-14}$ |
| 3 | p_n0_d15 | $\mathbf{1.80e-02 \pm 1.19e-04}$ | $\mathbf{8.00e-04 \pm 3.52e-07}$ | $1.02e-06 \pm 9.33e-13$ | $3.00e-07 \pm 4.86e-14$ |
| 4 | c_n0_d15 | $1.09e-04 \pm 1.67e-07$ | $5.66e-05 \pm 1.09e-07$ | $\mathbf{9.52e-07 \pm 8.05e-13}$ | $\mathbf{2.98e-07 \pm 8.76e-14}$ |
| 5 | p_n0_d30 | $\mathbf{3.42e-02 \pm 6.32e-04}$ | $\mathbf{1.55e-03 \pm 2.26e-06}$ | $1.00e-06 \pm 5.42e-13$ | $3.38e-07 \pm 3.94e-14$ |
| 6 | c_n0_d30 | $4.88e-05 \pm 6.92e-09$ | $7.63e-06 \pm 1.64e-10$ | $\mathbf{1.09e-06 \pm 9.07e-13}$ | $\mathbf{4.69e-07 \pm 1.26e-13}$ |
| 7 | p_n0_dco | $\mathbf{5.77e-06 \pm 5.96e-11}$ | $\mathbf{8.99e-07 \pm 1.58e-12}$ | $9.42e-07 \pm 4.52e-13$ | $2.57e-07 \pm 2.88e-14$ |
| 8 | c_n0_dco | $2.38e-05 \pm 5.47e-10$ | $3.78e-06 \pm 1.18e-11$ | $\mathbf{8.29e-07 \pm 4.56e-13}$ | $\mathbf{2.34e-07 \pm 2.07e-14}$ |
| 9 | p_n1_d00 | $\mathbf{1.13e-03 \pm 9.24e-07}$ | $\mathbf{1.50e-04 \pm 2.01e-08}$ | $5.68e-04 \pm 1.03e-07$ | $1.03e-04 \pm 8.17e-09$ |
| 10 | c_n1_d00 | $1.64e-02 \pm 2.24e-04$ | $2.88e-03 \pm 6.01e-06$ | $\mathbf{5.58e-04 \pm 1.65e-07}$ | $\mathbf{1.34e-04 \pm 1.30e-08}$ |
| 11 | p_n1_d15 | $\mathbf{1.80e-02 \pm 1.29e-04}$ | $8.39e-04 \pm 4.10e-07$ | $6.50e-04 \pm 1.89e-07$ | $1.02e-04 \pm 4.54e-09$ |
| 12 | c_n1_d15 | $3.78e-02 \pm 8.74e-04$ | $4.50e-03 \pm 2.16e-05$ | $\mathbf{5.91e-04 \pm 1.35e-07}$ | $\mathbf{1.20e-04 \pm 1.32e-08}$ |
| 13 | p_n1_d30 | $\mathbf{3.61e-02 \pm 6.15e-04}$ | $\mathbf{1.45e-03 \pm 2.32e-06}$ | $6.18e-04 \pm 1.21e-07$ | $1.05e-04 \pm 4.82e-09$ |
| 14 | c_n1_d30 | $2.19e-02 \pm 6.25e-04$ | $3.22e-03 \pm 1.12e-05$ | $\mathbf{6.28e-04 \pm 2.97e-07}$ | $\mathbf{1.17e-04 \pm 2.04e-08}$ |
| 15 | p_n1_dco | $\mathbf{1.33e-03 \pm 1.31e-06}$ | $\mathbf{2.24e-04 \pm 2.75e-08}$ | $5.20e-04 \pm 1.07e-07$ | $1.07e-04 \pm 4.69e-09$ |
| 16 | c_n1_dco | $2.21e-02 \pm 2.50e-04$ | $2.87e-03 \pm 5.26e-06$ | $\mathbf{5.83e-04 \pm 1.48e-07}$ | $\mathbf{9.18e-05 \pm 7.96e-09}$ |

### 6.2.2 Registration Results

The results presented in Table 6.3 show the benefits of using ICP as a downstream step for RTS registration algorithms. In all cases, we observed a decrease in error and uncertainty when using ICP. In addition, the proposed initialization method reduces the required user interaction for registration. For special setups, camera-based targeting can be avoided at all. However, the majority of the test sets result in a better initial registration when using the manual method, which was to be expected. In particular, we assumed an idealized setup, for which the targeting uncertainty was reduced to the sensor noise of the RTS. User dependent variations have not been analyzed. As-built discrepancies of CAD models have high influence on the initialization methods. Hence, expert knowledge and experience in measuring natural targets cannot be replaced. However, currently, the traditional method does not include ICP, while it is an integral step for our proposed method.

## 6.3 Discussion

In this chapter, we introduced a semi-automatic method for registration of RTS with respect to a CAD model. By using different methods for initial registration, we show the flexibility of the proposed algorithm. We explicitly avoided computationally complex methods like 3D mesh template matching or learning algorithms for 3D corner detection and pose estimation to limit the power consumption and run-time to an acceptable level.

We proposed a robust Manhattan-like corner estimation method for ICP initialization. Alternative methods that differ in the amount of user interaction and reliability of the initial registration, like single wall initialization or Manhattan-like structures with two walls, could be developed. Shape-priors could be extracted from the CAD selection. While this might be more robust in terms of detection, the added causal dependency would increase overall duration of the registration workflow. In the current work, the ordered point cloud of the initial measurements was interpreted as 3D polygon, and the interior angles of the polygon edges were used for clustering. Alternatives methods could be analyzed regarding the reliability of the initial clustering.

We used three point correspondences as reference registration method. One next step is to generalize point correspondences to point-and-direction correspondences to which covers a broader range of model registration methods. Furthermore, effects from systematic measurement errors like scale differences between current measurements and the CAD model can be analyzed or compensated.

The analysis of computer vision algorithms and user dependent variations are beyond the scope of this work. User studies with a physical setup could be used in addition to the proposed analysis. This additional insights and a more practical comparison of the proposed algorithms.

The proposed candidate-picking algorithm for measurement samples highly relies on the level of detail of the CAD model. Incorporating further information, such as saliency regions, could further decrease the uncertainty of the RTS registration. In particular, different sampling strategies on a spherical saliency map could be applied.

# 7 Conclusion

We think that the proposed assistance and simulation systems may have a significant impact and are a step towards intelligent RTS instruments. Especially for non-experts, small-size and mid-size companies, these methods can lower the entrance threshold for working with RTS and performing accurate, reliable and repeatable tasks. Using semi-automatic extraction of a simulator has proven to be a powerful tool to test applications, hardware configurations and environment influences in early design phases.

We tried to ensure the portability of the proposed workflows for a larger variability of devices by targeting commercially available RTS and reducing the hardware requirements to core functions of these instruments. To further lower the costs and effort for the transfer of knowledge to industry, we do not rely on multiple cameras or sensor data synchronization.

In the following, the achievements of the proposed methods are summarized, and personal opinions on trends and forecasts in this field are provided.

## 7.1 Particular Achievements

With the first workflow, we addressed targeting building corners manually. We showed that local approximations of the measured structures can lower the uncertainty and increase the reliability of measuring structures with at least one quasi-planar surface. The conducted user-study further indicates that non-expert users would prefer the proposed targeting methods, even though additional samples must be measured. The participants confirmed that the proposed user interface was intuitive, but a visual quality indicator of the expected targeting uncertainty would be beneficial.

With the second workflow we addressed the registration procedure of an RTS with respect to a polygonal CAD model in indoor environments. We kept the approach flexible by using a modular workflow and by splitting the procedure into coarse and fine registration. This allowed us to investigate different initialization methods and to analyze CAD-guided sampling heuristics for the refinement step. Our simulation results indicate that the ICP algorithm can be used for device registration as long as a sufficient initialization and careful sample-picking is provided. The latter is supported by a-priori information, which mainly consists of a potential visibility set of visible regions of the CAD model.

The proposed initialization method uses a corner with three quasi-perpendicular and quasi-planar surfaces. We consider this not only as one of many possible initialization methods, but also as first step toward scene understanding with sparse point sets of RTS measurements.

Apart from developer tests and some conceptual demos, the registration flow was only tested within the simulator. However, our research team currently is working on evaluations with laboratory conditions and with realistic setups. We also hypothesize that formalizing the coarse registration task as 2½D problem rather than as 3D problem would lead to better results as walls are usually approximately aligned with the gravity vector.

We believe that the developments mentioned above would not have been possible without the RTS prototyping and simulation system that we developed in early states of the project. The presented systematic model extracting approach for RTS kinematic simulation has proven to be particularly useful for prototypical devices. This allowed us to deal with incomplete drivers and documentations, to test concepts beyond device limitations, and to verify implementations during the complete development cycle in an efficient way.

However, it also should be mentioned that a prototyping framework with a driver abstraction layer can be a disadvantage in later design phases and will add additional effort to the transfer of knowledge to industry. We experienced that a more economic approach would be to use low-level communication protocols or generated mock interfaces of drivers to ensure the same API, which would allow easier integration into the existing infrastructure of companies.

## 7.2 Developments, Forecasts and Trends of RTS in the Construction Industry

A booming field of applications for RTS is the construction industry, where an increasing need for efficient information management, digital recordings and documentation can be observed. In 2015, Oxford Economics[1] and Global Construction Perspectives[2] published a study about prospects for the global construction industry [15]. The report predicts a worldwide market grow by 85% ($8 trillion) between 2015 and 2030. This trend also

[1]Oxford Economics is a leading company for business, economic and public policy advice and provides forecasts and analytical tools to international institutions, governments and blue-print companies [81].

[2]Global Construction Perspectives is an UK consulting company for forecasts and strategic issues about the global construction and engineering industry [39].

leads to an increasing need for managing construction projects more efficiently. One such trend is the concept of building information modeling (BIM). BIM defines standards and methods for optimizing design, execution and maintenance of construction projects. In particular, it centralizes design, coordination and documentation with the goal to reduce project costs, efforts and risks. BIM is standardized by ISO 19650-1:2018 and ISO 19650-2:2018 [53, 54] and built upon a centralized project database with continuous reports of the construction progress. Here, RTS are used in various construction phases, from setting out buildings, to regular recordings of the as-built state and documentations of as-built discrepancies, up to capturing structural building modifications at later states [59].

We think that this trend will impact the development of RTS and especially will increase the need for RTS in the mid-price range. We also expect standardized interfaces, simple-to-use software for non-experts, and RGB-D image sensors will become standard equipment for these instruments. The integration into CAD/BIM software will support measure-to-CAD approaches and will decrease the workload for daily recording routines.

Some products in this field are already commercially available and allow a forecast of upcoming feature set of these instruments. An example product in this field is the AutoCAD application *TachyCAD* from Faro Technologies [35], which supports CAD drawings on site by providing AutoCAD control and measurement functions for a wide range of different RTS.

We expect that a variety of comparable software will appear on the market and will gain in importance on construction sites. Associated therewith will RTS become increasingly autonomous in the near future.

## 7.3 Final Statements

The motivation of this work was to develop user-assistance systems for two typical RTS workflows.

The results of this work substantiate the benefits of alternating traditional measurement concepts by integrating ideas from other research fields, such as robotics, computer vision, virtual reality and augmented reality.

We believe that the growing construction market and the transition towards centralized management of building projects will increase the need for automation, efficiency and usability of RTS tasks. This forecast is endorsed by the increasing popularity of BIM concepts and by results of global market surveys [15].

We also understand that changes of established measurement methods in surveying and construction are slow and can only be done in small steps, as each alternation must be carefully tested, verified and often standardized. However, we believe that this field holds also great potential, and that the interest of users and companies in more automation will inspire further developments on one hand and will increase construction productivity and quality on the other hand.

# A Math Primer

In this chapter, we provide a selection of mathematical operations that are used frequently throughout this work or that are ambiguously defined in related literature. Note that the intention of this chapter is not to provide a comprehensive mathematical handbook, but rather to collect supplementary materials for ease of understanding and of the proposed workflows.

The interested reader is referred to following three books for a detailed collection of related mathematical and geometrical operations: Bartsch [12] provides comprehensive handbook of fundamental mathematical formulas, Hartley and Zisserman [44] excessively handle projective geometry and camera models, and Schneider and Eberly [94] discuss geometric solutions of common problems for the 2D and 3D space.

Unless otherwise stated, the operations discussed in this chapter can be found in the books mentioned above.

## A.1 Scalar Operations

**Sign Function** The sign function returns the sign of real value. Let $x$ be a scalar variable. In this work, the sign function $\text{sign}(x)$ is defined as

$$\text{sign}(x) = \begin{cases} -1 & x < 0 \\ 0 & x = 0 \\ 1 & \text{otherwise} \end{cases} \tag{A.1}$$

**Two-Argument Arctangent** The two-argument arctangent $\alpha = atan2(y, x)$ returns the signed rotation angle $\alpha$ between the x-axis and a vector $\mathbf{v} = [x\ y]^T$ in the Euclidean plane.

Let $\mathbf{v} = [x\ y]^T$ be a vector in the Euclidean plane, and $r$ be the length of this vector, according to

$$x = r\cos(\alpha) \quad y = r\sin(\alpha) \qquad r = ||\mathbf{v}|| \tag{A.2}$$

Then, the inverse trigonometric function $\arctan(\frac{y}{x})$ returns the correct angle for $x > 0$, but has a discontinuity at $x = 0$, and loses the quadrant information for $x < 0$ by dividing the input parameters.

The unique arc tangent value and the correct quadrant for the interval $\alpha \in (-\pi, \pi]$ can alternatively be calculated using $\alpha = atan2(y, x)$. One possible definition of the two-argument arctangent is given by

$$\alpha = atan2(y,x) = \begin{cases} arctan\left(\frac{y}{x}\right) & x > 0 \\ arctan\left(\frac{y}{x}\right) + \pi & x < 0 \text{ and } y \geq 0 \\ arctan\left(\frac{y}{x}\right) - \pi & x < 0 \text{ and } y < 0 \\ +\frac{\pi}{2} & x = 0 \text{ and } y > 0 \\ -\frac{\pi}{2} & x = 0 \text{ and } y < 0 \\ \text{undefined} & x = 0 \text{ and } y = 0 \end{cases} \tag{A.3}$$

## A.2 Linear Algebra

### A.2.1 Matrix Operations

**Rotation** In this work, the homogeneous rotation matrices are defined as following[1]:

$$\mathbf{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{R}_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{R}_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.4}$$

We represent an Euler rotation with a 3D rotation matrix according to

$$\mathbf{R} = \mathbf{R}_z(\gamma)\mathbf{R}_y(\beta)\mathbf{R}_x(\alpha) \tag{A.5}$$

where $(\alpha, \beta, \gamma)$ are the Euler angles as described by Schneider and Eberly [94].

**Translation** The translation matrices used in this work are given by:

$$\mathbf{T}_x(x) = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{T}_y(y) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{T}_z(z) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.6}$$

[1]For the economy of notation, $\mathbf{R}_x$, $\mathbf{R}_y$ and $\mathbf{R}_z$ can represent homogeneous or Euclidean rotation matrices for the 3D space throughout this work.

### A.2.2 3D Point Clouds

**Center of Mass** Given a collection of 3D points $S_p = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$, the center of mass $\bar{\mathbf{x}}$ is given by

$$\bar{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i \tag{A.7}$$

Further point cloud problems are discussed in Appendix A.3.

### A.2.3 3D Primitives

**Plane** A plane $\mathbf{p}$ in 3D can be defined by a homogeneous vector according to

$$\mathbf{p} = \begin{bmatrix} \mathbf{n} \\ -\mathbf{n}^T \cdot \bar{\mathbf{x}} \end{bmatrix} \qquad ||\mathbf{n}|| = 1 \tag{A.8}$$

where $\mathbf{n} = [n_x\ n_y\ n_z]^T$ is the normalized plane normal, and $\bar{\mathbf{x}}$ is a point on the plane. Given three or more points on the plane, $\bar{\mathbf{x}}$ the best estimate in the least square sense is given by the center of mass according to A.7.

**Ray** The parametrized form of a $n$ dimensional ray is given by

$$\mathcal{R}(l) = \mathbf{r}_0 + l\mathbf{d}_0 \tag{A.9}$$

where $\mathbf{r}_0$ is the ray start point, $\mathbf{d}_0$ is the normalized ray direction, and the parameter $l$ is the distance between the start point and the ray endpoint.

**Line** A line in n dimension can be defined by two points on the line $(\mathbf{x}_1, \mathbf{x}_2)$. The parameterized ray equation Eqn. A.9 can also be used to define a $n$ dimensional line $\mathcal{L}(l)$ according to

$$\mathcal{L}(l) = \mathbf{x}_1 + l\frac{\mathbf{x}_2 - \mathbf{x}_1}{||\mathbf{x}_2 - \mathbf{x}_1||} \tag{A.10}$$

where $l$ is the distance between the two points. The normalization ensures that the parameter $l$ is true to scale.

## A.3 SVD

Hartley and Zisserman [44] describe SVD as one of the most useful matrix decompositions for numerical computations, with applications in fields like solving linear equations, principal component analysis, least square minimization, rank estimation, point set alignment, or nearest orthogonal matrix estimation.

SVD is a generalization of the spectral decomposition of a matrix. The operator factorizes a matrix into three matrices that consists of terms of eigenvalues and eigenvectors.

The SVD of a $M \times N$ *matrix* $\tilde{\mathbf{A}}$ is defined as

$$SVD(\tilde{\mathbf{A}}) = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T \tag{A.11}$$

where $\mathbf{U}$ is a $M \times M$ matrix, $\mathbf{S}$ is a $M \times N$ matrix, $\mathbf{V}$ is a $N \times N$ matrix. The matrix $\mathbf{S}$ is a diagonal matrix with the singular values of $\tilde{\mathbf{A}}$ as its entries. A common convention is to sort the entries of $\mathbf{S}$ in descending order, whereby the entries are non-negative, real-valued numbers. Both, $\mathbf{U}$ and $\mathbf{V^T}$ are unitary, where the columns of $\mathbf{U}$ and $\mathbf{V^T}$ are orthogonal bases, respectively.

A detailed description of the algorithm along with solutions to various practical fitting problems can be found in the book by Hartley and Zisserman [44] and in related relevant literature.

In the following, we briefly discuss selected SVD applications that are used in this work.

### A.3.1 Fitting a Plane to a Point Cloud Using SVD

A plane normal $\mathbf{n}$ can be fitted to the point cloud $S_p$ using SVD. Let $\tilde{\mathbf{A}}$ be the $3 \times N$ matrix containing the normalized and centered measurement samples according to

$$\mathbf{A} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_N \end{bmatrix} \tag{A.12}$$

$$\bar{\mathbf{A}} = \begin{bmatrix} (\mathbf{x}_1 - \bar{\mathbf{x}}) & (\mathbf{x}_2 - \bar{\mathbf{x}}) & \dots & (\mathbf{x}_N - \bar{\mathbf{x}}) \end{bmatrix} \tag{A.13}$$

$$\tilde{\mathbf{A}} = \frac{1}{k}\bar{\mathbf{A}} \qquad k = \max_{a_{ij} \in \bar{\mathbf{A}}} |a_{ij}| \tag{A.14}$$

In this case, the normalization $k$ is the absolute maximum value of all elements in $\bar{\mathbf{A}}$. A vector parallel to the plane normal $\mathbf{n}$ is given by the eigenvector of $\tilde{\mathbf{A}}$, which corresponds to the smallest eigenvalue, which is simply the last column $\mathbf{u}_3$ of $\mathbf{U} = [\mathbf{u}_1\ \mathbf{u}_2\ \mathbf{u}_3]$. The plane $\mathbf{p}$ is then fully defined by Eqn. A.8, where $\mathbf{n} = \frac{\mathbf{u}_3}{||\mathbf{u}_3||}$ is the plane normal, and the center of mass $\bar{\mathbf{x}}$ of the point cloud is given in Eqn. A.7.

### A.3.2 Fitting the Relative Euclidean Transformation between Two Ordered Point Sets Using SVD

Estimating 3D rigid transformation between two ordered point sets has been studied extensively in literature [8, 29, 46, 47]. In this work, we follow the SVD approach as proposed by Arun et al. [8], which requires known point correspondences.

Let $S_a = (\mathbf{a}_1, \mathbf{a}_2 ... \mathbf{a}_N)$ and $S_b = (\mathbf{b}_1, \mathbf{b}_2 ... \mathbf{b}_N)$ be two ordered point sets, where the point correspondences are given by the element indices. Then, at least three point correspondences are needed for determining the Euclidean transformation.

In general, we want to find the best Euclidean transformation between the ordered point sets in the least squares sense:

$$(\mathbf{R}, \mathbf{t}) = F(S_a, S_b) = \arg\min_{\mathbf{R},\mathbf{t}} \sum_{i=1}^{N} ||\mathbf{R}\mathbf{a}_i + \mathbf{t} - \mathbf{b}_i||^2 \tag{A.15}$$

The centroids $\bar{a}$ and $\bar{b}$ of of the point sets are given by

$$\bar{\mathbf{a}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{a}_i \qquad \bar{\mathbf{b}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{b}_i \tag{A.16}$$

The cross-covariance matrix of the two point sets is given by

$$\mathbf{H} = (\mathbf{S}_a - \bar{\mathbf{a}})(\mathbf{S}_b - \bar{\mathbf{b}})^T \tag{A.17}$$

where the measurement matrices $\mathbf{S}_a$ and $\mathbf{S}_b$ contains the stacked points of the two ordered point sets according to

$$\mathbf{S}_a = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & ... & \mathbf{a}_N \end{bmatrix}^T \qquad \mathbf{S}_b = \begin{bmatrix} \mathbf{b}_1 & \mathbf{b}_2 & ... & \mathbf{b}_N \end{bmatrix}^T \tag{A.18}$$

Let the SVD of the cross-covariance matrix $\mathbf{H}$ be $SVD(\mathbf{H}) = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T$ according to Eqn. A.11. Then, the rotation $\mathbf{R}$ can be calculated by

$$\mathbf{R} = \mathbf{V} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & det(\mathbf{V}\mathbf{U}^T) \end{bmatrix} \mathbf{U}^T \tag{A.19}$$

Finding the optimal rotation $\mathbf{R}$ is also known as *Kabsch algorithm* [61].

The optimal translation $\mathbf{t}$ is given by

$$\mathbf{t} = -\mathbf{R}\bar{\mathbf{a}} + \bar{\mathbf{b}}. \tag{A.20}$$

Finally, the rigid transform $\mathbf{M}$ that defines the transformation of point cloud $S_a$ to point cloud $S_b$ is given by

$$(\mathbf{R}, \mathbf{t}) = F(S_a, S_b) \qquad \mathbf{M} = \left[\begin{array}{ccc|c} & \mathbf{R} & & \mathbf{t} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{A.21}$$

### A.3.3 Fitting Relative Euclidean Transformation Between Two Coordinate Frames Using SVD

Let $(\mathbf{o}_0, \mathbf{x}_0, \mathbf{y}_0, \mathbf{z}_0)$ and $(\mathbf{o}_1, \mathbf{x}_1, \mathbf{y}_1, \mathbf{z}_1)$ be two local coordinate frames in Euclidean space. Then, the Euclidean pose of frame $(\mathbf{o}_1, \mathbf{x}_1, \mathbf{y}_1, \mathbf{z}_1)$ with respect to frame $(\mathbf{o}_0, \mathbf{x}_0, \mathbf{y}_0, \mathbf{z}_0)$ is given by a six DOF Euclidean transformation $\mathbf{M}$.

This problem can be reduced to finding the 3D rigid transformations between two ordered point sets as described in Appendix A.3.2.

In particular, by converting the origin and the direction of the axis to corresponding ordered sets of points, the problem can be written as:

$$\left.\begin{array}{ll} S_a = (\mathbf{c}_0 + \mathbf{x}_0, \mathbf{c}_0 + \mathbf{y}_0, \mathbf{c}_0 + \mathbf{z}_0) & S_b = (\mathbf{c}_1 + \mathbf{x}_1, \mathbf{c}_1 + \mathbf{y}_1, \mathbf{c}_1 + \mathbf{z}_1) \\ (\mathbf{R}, \mathbf{t}) = F(S_a, S_b) & \mathbf{M} = \left[\begin{array}{ccc|c} & \mathbf{R} & & \mathbf{t} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \end{array}\right\} \tag{A.22}$$

where the rigid transform $\mathbf{M}$ describes the pose of frame $(\mathbf{o}_1, \mathbf{x}_1, \mathbf{y}_1, \mathbf{z}_1)$ with respect to frame $(\mathbf{o}_0, \mathbf{x}_0, \mathbf{y}_0, \mathbf{z}_0)$ and can be calculated according to Eqn. A.15.

## A.4 RTS Camera Projection Matrices

Understanding camera properties and calibration methods is crucial for developing visual RTS algorithms. The book by Hartley and Zisserman [44] provide a detailed discussion about cameras, different camera models, projection matrices and matrix properties. The authors also discuss camera calibration and explain photogrammetric reconstruction. Ehrhart further describes camera calibration and related models in thecontext of RTS instruments [32].

As the authors mentioned above cover cameras sufficiently for developing visual RTS algorithms, this section provides only the basics for understanding the equations used in this work.

### A.4.1 Camera Projection Matrix

Given the camera intrinsics $\mathbf{K}$, projection of 3D points to image space can be written as [44]:

$$\hat{\mathbf{u}}_i = \mathbf{P}_{\text{CAM}} \mathbf{M}_{B,C}^{-1} \hat{\mathbf{x}}_i \qquad \mathbf{P}_{\text{CAM}} = [\mathbf{K}|\mathbf{0}] \qquad \mathbf{K} = \begin{bmatrix} f_u & s & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix} \tag{A.23}$$

where $\hat{\mathbf{u}}_i$ is a homogeneous point on the image plane, $\hat{\mathbf{x}}_i$ is a homogeneous point in space $B$, $s$ is a skew factor, $(f_u, f_v)$ describes the focal lengths and $\mathbf{c} = [c_u \ c_v]^T$ describe the 2D principal point offset; $\mathbf{M}_{B,C}$ is the camera pose, hence $\mathbf{M}_{B,C}^{-1}$ transforms a point from the base frame $B$ to the camera frame $C$. Note that this model does not consider distortion parameters.

If the intrinsic camera parameters are not accessible by the API of an RTS instrument, the parameters can be estimated using a standard camera calibration toolbox like *OpenCV* [19]. Details about different camera calibration methods are provided in the book by Hartley and Zisserman [44] and in related relevant literature.

### A.4.2 Point Reprojection From Image to World

The decomposition of the projection matrix $\mathbf{P} = [\mathbf{M}|\mathbf{p}_4]$ into the $3 \times 3$ matrix $\mathbf{M}$ and the $3 \times 1$ column vector $\mathbf{p}_4$ is used for view ray calculation of finite cameras: A numerical robust back-projection of a image coordinate $\hat{\mathbf{u}}$ to the view ray $X_T(\hat{\mathbf{u}}, \mu)$ with metric parametrization of the ray length $\mu$ is given by Klug et al. [66]:

$$X_T(\hat{\mathbf{u}}, \mu) = \frac{\mu}{||\mathbf{M}^{-1} \cdot \hat{\mathbf{u}}||} \begin{bmatrix} \mathbf{M}^{-1} \cdot \hat{\mathbf{u}} \\ 0 \end{bmatrix} + \begin{bmatrix} -\mathbf{M}^{-1} \cdot \mathbf{p}_4 \\ 1 \end{bmatrix} \tag{A.24}$$

Compared to the reprojection methods proposed by Hartley and Zisserman [44], this equation additionally allows calculating an Euclidean 3D point rather than the Euclidean 3D ray only, as long as depth information is stored along with the image data.

B

# Implementation Details of the RTS Software Development System

In the following, we provide supplementary material for the in chapter 4 proposed prototyping and simulation system of the exemplary RTS. In particular, this chapter covers the software architecture and implementation details of the simulator shown in Fig. 4.6, Fig. 5.6 and Fig. B.1. Furthermore, we provide a description of the custom code generators for interconnecting applications, physical devices and the simulator.

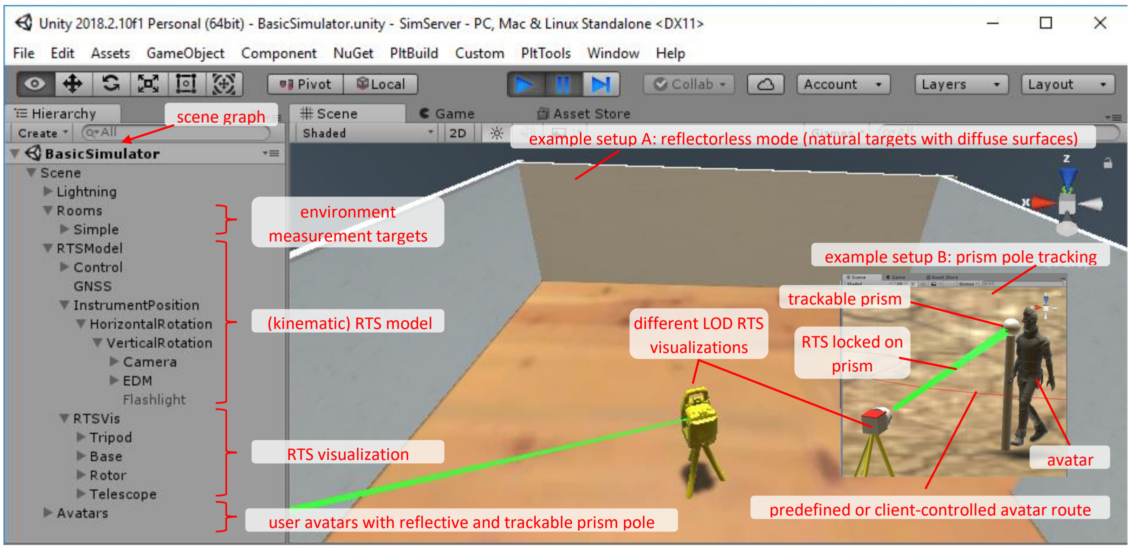

Unity 2018.2.10f1 Personal (64bit) - BasicSimulator.unity - SimServer - PC, Mac & Linux Standalone <DX11>
File Edit Assets GameObject Component NuGet PltBuild Custom PltTools Window Help
Pivot Local Collab Account Layers Layout
Hierarchy Create All
Scene Shaded 2D Game Asset Store
BasicSimulator
Scene
Lightning
Rooms
Simple
RTSModel
Control
GNSS
InstrumentPosition
HorizontalRotation
VerticalRotation
Camera
EDM
Flashlight
RTSVis
Tripod
Base
Rotor
Telescope
Avatars
scene graph
environment measurement targets
(kinematic) RTS model
RTS visualization
user avatars with reflective and trackable prism pole
example setup A: reflectorless mode (natural targets with diffuse surfaces)
example setup B: prism pole tracking
trackable prism
different LOD RTS visualizations
RTS locked on prism
avatar
predefined or client-controlled avatar route

Figure B.1: Unity3D-based RTS simulator used in this work. Two different targeting modes are shown, namely reflectorless mode (example setup A) and prism tracking mode (example setup B). Software prototypes can communicate with either the simulator or the physical instrument by means of a automatically generated remote procedure call modules on top of the gRPC library [41].

## B.1 Motivation

The implementation of the proposed ecosystem for heterogeneous RTS software design and simulation was motivated by following three observations:

**Effort and Risk of Repeating Test Data Recordings** Creating suitable datasets for exploratory research can be expensive. Simulations during conceptual and preliminary system design phases help identifying key factors for success and failure, measurement properties and environmental influences. We experienced that a flexible ecosystem for simulation and prototyping can help to implement a ”first-time-right” strategy for database creation and to lower the risk of repeating time-consuming database steps, such as data recording, cleaning and labeling.

**Benefits of Mockups and Early User Test Runs** Mockups and stubs are useful for collecting user feedback about interfaces and workflows during early design phases, for detecting biased designer assumptions and for eliminating blind spots in system concepts. Furthermore, simulations of interactive algorithms and measurement setups beyond actual hardware limitations allow efficient and extensive testing and also identifying important properties for successor instruments. We also experienced that simulations can help to increase the competitiveness of research groups and companies: Novel methods, processes and machines can be simulated efficiency and invention applications can be filed before extensive physical experiments are conducted.

**Driver and Software Limitations** We experienced driver and firmware limitations when working with hardware and software prototypes. This is a common scenario when using leading-edge instruments, especially when the particular device is not officially released yet, and its hardware or software is still under development. A common release strategy in industry is to first deliver robust core functionalities and to add additional features that might allow device exploration beyond existing use cases later. Unfortunately, this can delay experiments and reduce the otherwise impressive range of research possibilities with such instruments, especially when the requested features are postponement or dropped due to their low priority for actual products. Another observation was that combining existing software building blocks, multiple platforms or different programming languages can increase the efficiency for prototyping, but also does require an ecosystem for heterogeneous RTS software development. For example, in this work, we experimented with combinations of mobile devices, desktop computers, various operation systems and different programming languages.

While existing solutions like the ROS library [82] handle some of the problems mentioned above to a certain extend, we are interested in a communication middleware that has a steeper learning curve, supports device servers for different operation systems, and provides a more transparent function call layer. Additionally, the proposed system provides the possibility to completely remove the communication library during later development states by porting all building blocks to the same runtime engine.

The following sections describe our software architecture, which allows combining existing building blocks to RTS workflows and software prototypes in an efficient way.

## B.2 Software System Architecture

**Device Drivers** The prototype of the instrument we used in this work was shipped with an pre-compiled driver and a software development kit (SDK). Both, driver and SDK are managed code assemblies, written in C# and executed with the Windows .NET Framework[1].

The integration of pre-complied and managed code assemblies into existing software becomes problematic and usually requires process interoperability if different runtime engines are involved. In this work, we use gRPC [41] as IPC library to avoid restricting software prototypes and related building blocks to the runtime engine and operation system of the driver. This concept allows combining building blocks and processor platforms with different operation systems, word sizes, and programming languages. We also experienced that the applied object-oriented design and software design patterns increase the maintainability of the system. If not mentioned differently, the software design patterns stated in this section refer to the book by Gamma et al. [38].

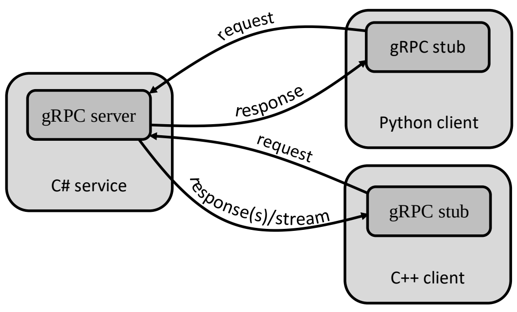
**System Architecture** The system architecture of our prototyping framework follows a *server-client software model*, which Reese [85] describes as *distributed application structural pattern*. First, we collect and encapsulate a subset of the RTS SDK into a common class and then convert this class to a device server. Method prototypes for external function calls are simplified using the *facade pattern*: Input and output parameters for server function calls are limited to single instances or lists of standard datatypes and enumerations by convention. Special language constructs like generic types, function overloading or a variable argument count are not allowed.

Accordingly, a client provides access to server functions and contains the actual program logic of an application.

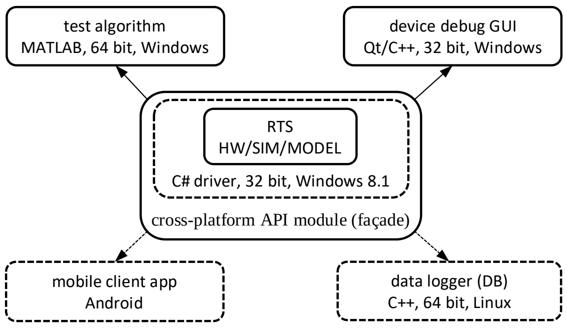
**Data Communication** The communication between server and clients in our system is based on the gRPC library, which implements remote procedure call (RPC) and follows the *request-response message-passing pattern*. The library recommends a *microservice architecture* and provides tools for generating cross-platform subs and bindings for servers and clients [41]. Figure B.2a and Figure B.2b show the similarities between the recommended

[1]The Windows .NET Framework is a Common Language Runtime (CLR) engine for the Windows operation system [55, 78].

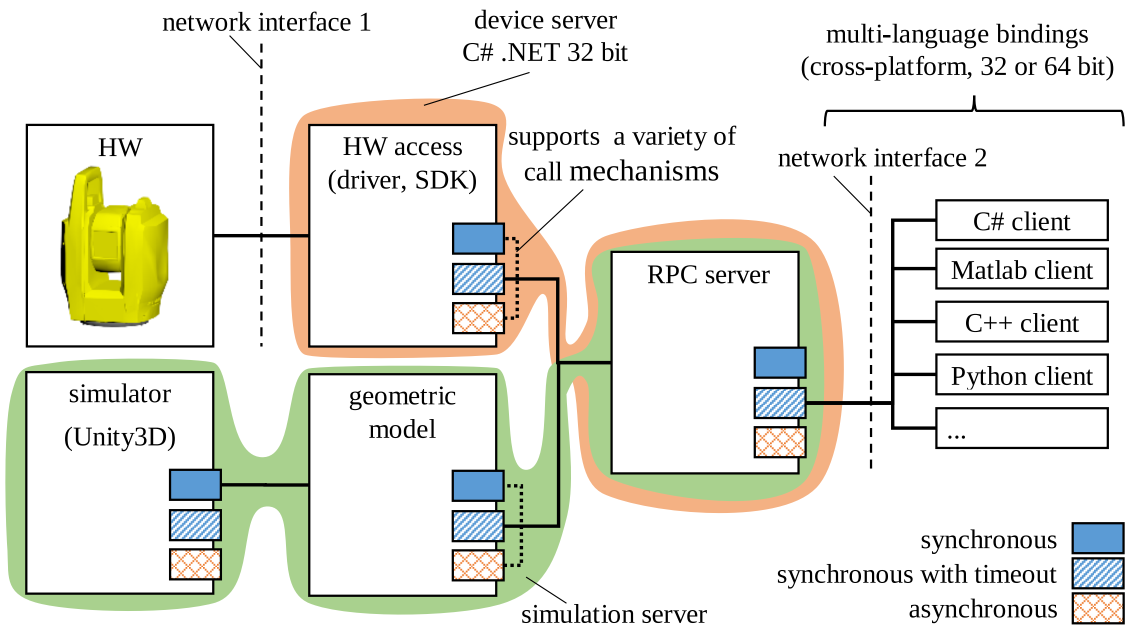
server-client model of gRPC and the desired software architecture for the prototyping system. We extend the library by tools for automatically generating the required API and messages. For this purpose, we use type introspection and customized, template-based code generators.



(a) server-client model of gRPC



(b) server-client model for RTS driver RPC



(c) detailed server-client model for prototyping system

Figure B.2: gRPC and RPC server-client models. The gRPC server-client model for microservices (a) is suitable for the desired RTS prototyping architecture (b). The system block design for the RTS prototyping framework (c) shows the interconnections between applications, simulator and the physical device. The instrument used for this work has a built-in network interface and supports wireless communication between driver and hardware. Both, driver and SDK are available for the Windows C# .NET Framework, pre-compiled for 32 bit architectures.

### B.2.1 Software Modules

Figure B.2c shows a detailed block diagram of the prototyping framework for the exemplary RTS. The individual blocks are described below.

**Hardware Access Module** The *hardware access module* is built upon the C# SDK of the exemplary RTS. It encapsulates the SDK functions into a single class, and defines public methods with simple function prototypes. RPC functions are marked by attributes and converted into an intermediate format for client and server code generation. This module can be executed standalone for test purposes or embedded in the device server for normal operation. We use template-based code-generators for automatic extraction of communication models, multi-language bindings and simulator stubs. In particular, we restrict the scope to basic function prototypes and data types and extend the tools of the gRPC library to produce ready for use clients and servers.

**Geometric Model and Simulation Module** The *geometric model and simulator module* implements the simple RTS position kinematics as presented in chapter 4. These two functions can be separated into two blocks as shown in Figure B.2c. This allows a single source implementation for simulation, data conversion and mockup tests, with the option to disable the rendering engine if not needed. Compared to a single simulation module, where everything is implemented in Unity3D, the separation into two modules is a more accurate picture of the reality as many SDK functions are available offline. However, this also leads to an increased software complexity.

**RPC Server Module** The *RPC server* provides a common access point for the simulation server and the device server. More precisely, both servers implement the same API, hence are interchangeable. The majority of the server functions are provided as *unary RPC*, where a client sends a single request and waits for the response, much like when using an ordinary function call [41]. Simultaneous requests of different clients are queued on the server-side. In addition, we support *server-side streaming RPC*, where one client sends a request and receives multiple responses in form of a stream[2].

**Client Modules** A client implements the access to the device or simulation server and the actual program logic of an RTS software prototype. The functions for client-server communication are generated automatically, but the program logic must be added manually, usually by means of class inheritance or composition [38]. Figure B.3 shows the communication stack for RPC and indicates generated and hand-crafted layers. Our system

[2]Currently, our system supports server-side streaming to C++, C# and Python clients, but not to Matlab. An exception is the image stream, for which Motion JPEG over a separate HTTP port is one of several streaming options. Streaming can also be emulated by running multiple polling clients, each executed in a separate thread within the client application. However, this is considered as inefficient and should be avoided.

contains templates for generating ready to use clients for C++, C#, Matlab, Unity3D and Python. Additionally, gRPC stubs and functions can be generated for all programming languages that are supported by the library [41].

To handle incompatibilities between generated C# clients and an existing C# SDK, one can generate unmanaged C++ clients with our system and add high-level C# bindings with the Simplified Wrapper and Interface Generator (SWIG) [13]. For example, we successfully applied this approach for accessing the exemplary RTS with Unity3D from mobile clients and from within AutoCAD.

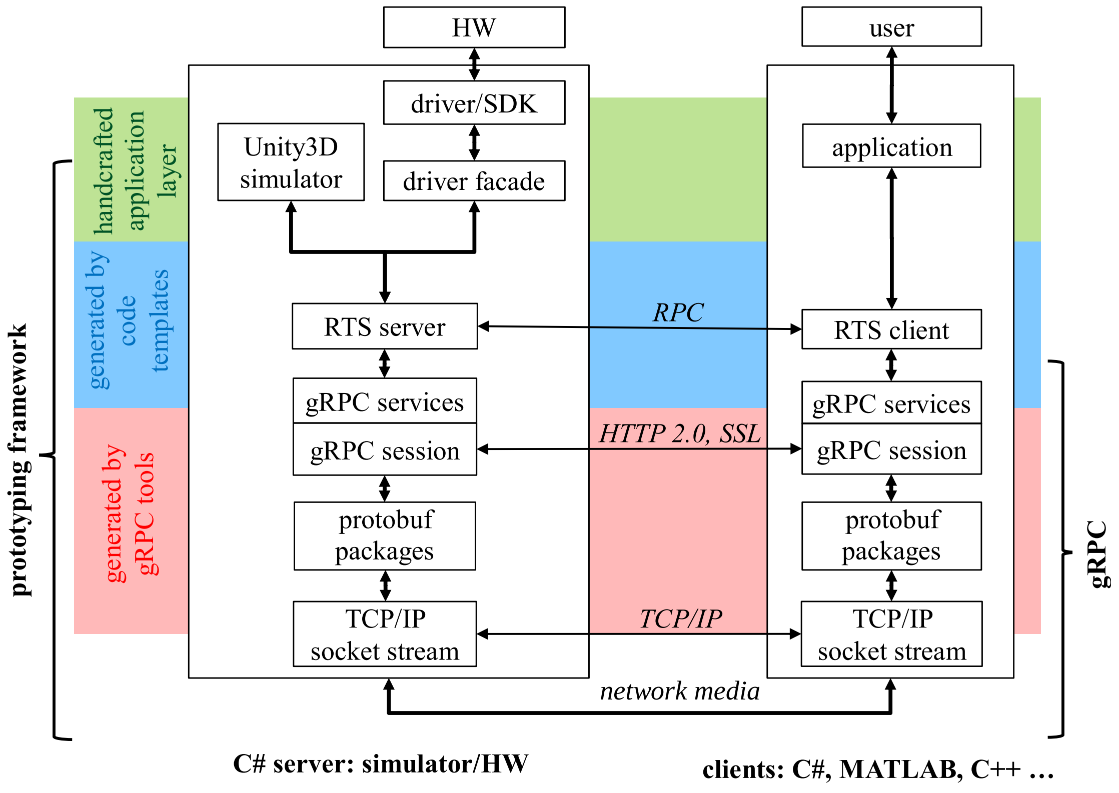


Figure B.3: Generated and handcrafted layers of the communication stack for RPC.

## B.3 RPC Code Generation

The code generation for ready to use clients and server subs can be broken down into following three steps:

First, relevant SDK functions are encapsulated into the driver access module, RPC methods are selected and marked by attributes as described in the previous section.

Then, the relevant information is stored as an intermediate representation for further processing. In particular, we use code introspection[3] and a top–down traversal of the driver

[3]Provided by C# code reflection.

access module to transform the class into an *abstract syntax tree*. Note that the generated intermediate structure does not contain a complete abstract syntax tree as described by Aho et al. [1], but only the relevant tree nodes shown in Fig. B.4.

Finally, we convert an existing gRPC client code example into a code generation template for the selected target language. Hereby, each nodes of the abstract syntax tree shown in Fig. B.4 represents a code snippet in the existing gRPC client code example, which are generalized for further processing. Each template contains a generator counterpart that takes the abstract syntax tree as input, concatenates code snippets to a complete implementation, and repeats insertions like methods or parameters where needed. Note that the first two steps of the workflow must be repeated for each client, while implementing the code generator is done only once for each target generator.

Figure B.5 shows the workflow for generalizing a standard gRPC client example to a code generator. The client source code shown in the figure is part of the gRPC library [41].

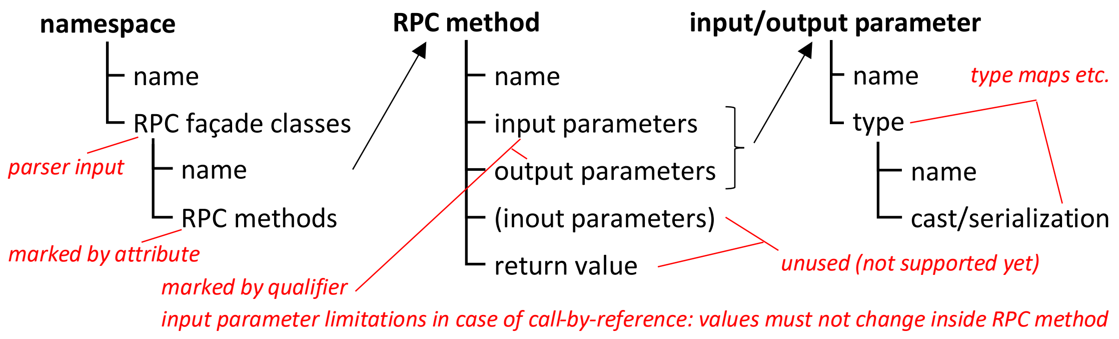
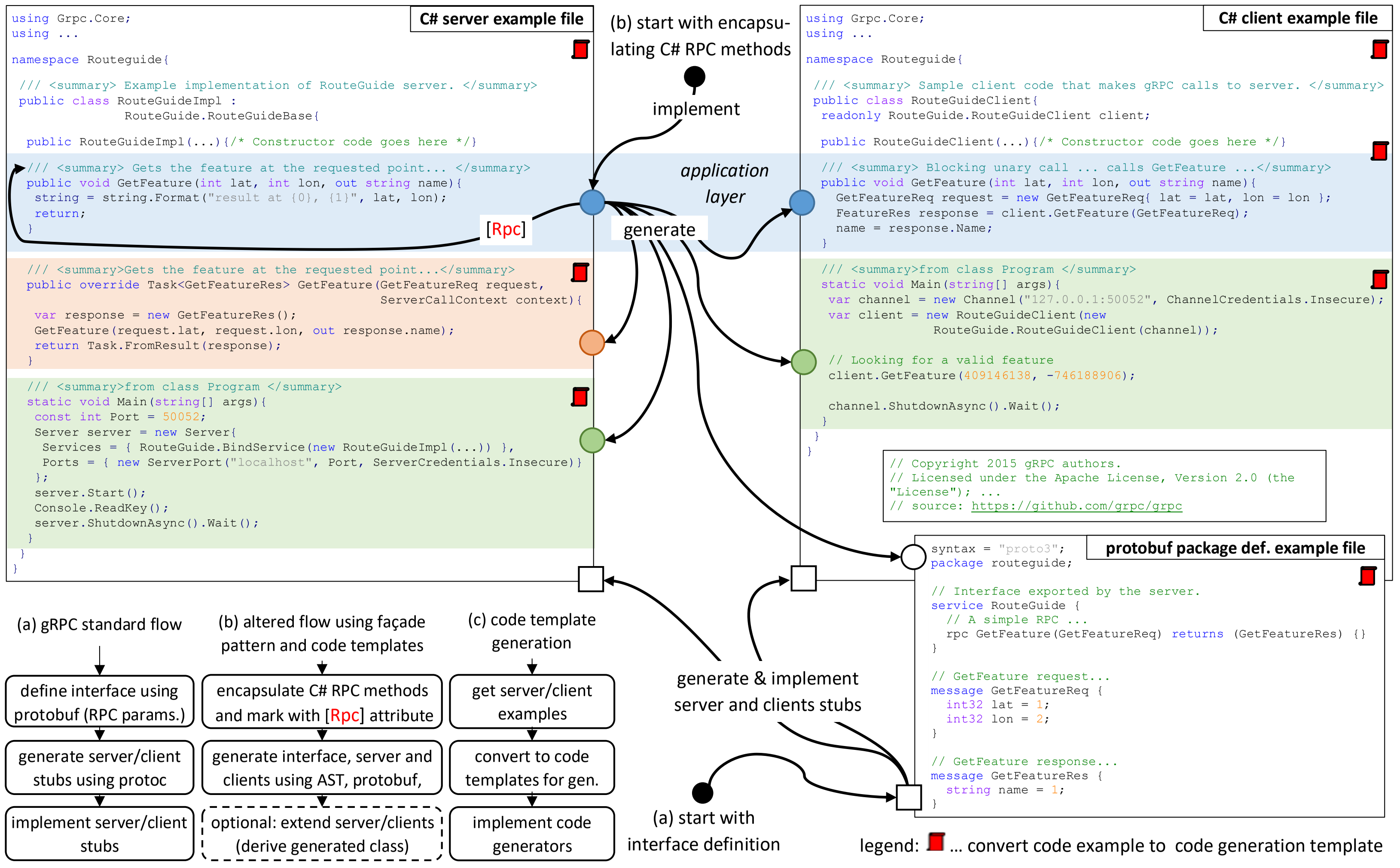


Figure B.4: Abstract syntax tree for code generation. In this work, we want to publish a C# driver as RPC service. By design rule, each service provider is encapsulated in a separate class, and each driver or SDK function call is encapsulated in a class method. The function prototype of these methods have certain limitations to reduce the RPC complexity (Gang of Four (GoF) facade pattern). The C# attribute *Rpc* is used to mark selected methods for publishing. Alternatively, the class itself can be marked to publish all public methods. The parser uses code reflection to convert the RPC class into an intermediate representation. Google Protocol Buffer (PROTOBUF) packages, clients and additional server stubs are then generated by template-based code generation. Note that each entry in the tree view has a separate template with placeholders in the root template.

## B.4 Unity3D as Simulator Frontend

Using a 3D rendering engine as frontend for real-time hardware simulation allows the integration of hardware properties, firmware behavior, environmental influences and timing constraints. Similar to Hu and Meng [49], Mattingly et al. [73] and Meng et al. [77], we use


C# server example file

```
using Grpc.Core;
using ...

namespace Routeguide{

 /// <summary> Example implementation of RouteGuide server. </summary>
 public class RouteGuideImpl :
               RouteGuide.RouteGuideBase{

  public RouteGuideImpl(...){/* Constructor code goes here */}

  /// <summary> Gets the feature at the requested point... </summary>
  public void GetFeature(int lat, int lon, out string name){
   string = string.Format("result at {0}, {1}", lat, lon);
   return;
  }

  /// <summary>Gets the feature at the requested point...</summary>
  public override Task<GetFeatureRes> GetFeature(GetFeatureReq request,
                                         ServerCallContext context){
   var response = new GetFeatureRes();
   GetFeature(request.lat, request.lon, out response.name);
   return Task.FromResult(response);
  }

  /// <summary>from class Program </summary>
  static void Main(string[] args){
   const int Port = 50052;
   Server server = new Server{
    Services = { RouteGuide.BindService(new RouteGuideImpl(...)) },
    Ports = { new ServerPort("localhost", Port, ServerCredentials.Insecure)}
   };
   server.Start();
   Console.ReadKey();
   server.ShutdownAsync().Wait();
  }
 }
}
```

C# client example file

```
using Grpc.Core;
using ...

namespace Routeguide{

 /// <summary> Sample client code that makes gRPC calls to server. </summary>
 public class RouteGuideClient{
  readonly RouteGuide.RouteGuideClient client;

  public RouteGuideClient(...){/* Constructor code goes here */}

  /// <summary> Blocking unary call ... calls GetFeature ...</summary>
  public void GetFeature(int lat, int lon, out string name){
    GetFeatureReq request = new GetFeatureReq{ lat = lat, lon = lon };
    FeatureRes response = client.GetFeature(GetFeatureReq);
    name = response.Name;
  }

  /// <summary>from class Program </summary>
  static void Main(string[] args){
   var channel = new Channel("127.0.0.1:50052", ChannelCredentials.Insecure);
   var client = new RouteGuideClient(new
                  RouteGuide.RouteGuideClient(channel));

   // Looking for a valid feature
   client.GetFeature(409146138, -746188906);

   channel.ShutdownAsync().Wait();
  }
 }
}
```

```
// Copyright 2015 gRPC authors.
// Licensed under the Apache License, Version 2.0 (the
"License"); ...
// source: https://github.com/grpc/grpc
```

protobuf package def. example file

```
syntax = "proto3";
package routeguide;

// Interface exported by the server.
service RouteGuide {
  // A simple RPC ...
  rpc GetFeature(GetFeatureReq) returns (GetFeatureRes) {}
}

// GetFeature request...
message GetFeatureReq {
  int32 lat = 1;
  int32 lon = 2;
}

// GetFeature response...
message GetFeatureRes {
  string name = 1;
}
```

[Rpc]
(b) start with encapsulating C# RPC methods
implement
application layer
generate
generate & implement server and clients stubs
(a) start with interface definition
legend: 🟥 ... convert code example to code generation template
(a) gRPC standard flow
define interface using protobuf (RPC params.)
generate server/client stubs using protoc
implement server/client stubs
(b) altered flow using façade pattern and code templates
encapsulate C# RPC methods and mark with [Rpc] attribute
generate interface, server and clients using AST, protobuf,
optional: extend server/clients (derive generated class)
(c) code template generation
get server/client examples
convert to code templates for gen.
implement code generators

Figure B.5: Workflows for implementing clients and servers and for generalizing a standard gRPC client code examples to code generators. (a) standard gRPC workflow, where messages are defines first, and server and client are manually implemented later. (b) our proposed workflow, where the hardware access module is implemented first, and server and clients are automatically generated later. (c) proposed workflow for generalizing example client code to code generator templates. The underlying client source code is included in the gRPC library [41].

Unity3D [105] for simulating the behavior of RTS sensors and actuators and the interaction with the environment.

In this section, we provide a description of the Unity3D frontend for the exemplary RTS simulator. In particular, we focus on software and implementation details, as the mathematical formalization of the modeled system have already been discussed in chapter 4.

### B.4.1 Memory Layout, Data Flow and Synchronization

**Processing Loops** The architecture of a complete RTS system usually have multiple decoupled processing loops.

On the application side, smooth user interaction is desired. Interactive applications are *event-driven* programs that react on user interactions, such as mouse clicks, key presses or touchpad sensors. An *event handler* is responsible for managing and processing messages and events.

Streamed data can be handled synchronously in the main thread by using polling mechanisms, asynchronously by using function callbacks for received packages, or in background threads by multiple client instances. For example, we split user-triggered measurements and updates from video-streams to ensure smooth user interaction.

When using the device server, an significant latency must be expected between application requests and response. With our simulation system, requests are written immediately to the *state request* or *control register*.

**Simulator Updates** The main loop of the game engine constantly reads the control register values, updates the scene simulation, and writes the scene state and simulation results to the *state* or *data register*. The game engine continuously updates of the internal states of the simulator and streaming data, such as video images, angles, distances, user avatars, and some kinematic and dynamic processes. Compared to the physical instrument, the simulator has significant lower communication and measurement latency. However, correct system behavior requires data synchronization for both, instrument and simulator.

The simulation server updates the state of the RTS model in each rendering pass and supports variable and fixed frame rates. While the communication module ensures only low-level synchronization of function calls, we manually applied concurrent computing techniques on top of the generated code where necessary.

**Client Updates and Data Streaming** Similar to the original SDK, the system supports client-side RPC, polling of camera frames and simple frame streaming. Similarly, continuous angle and distance measurements and prism pole tracking are supported. Besides, unbuffered angle, distance and image frames are stored separately in the data register; buffered and unbuffered data can be accessed simultaneously by different clients.

Note that a single application can contain multiple communication clients, which run asynchronous in different threads.

For example, an interactive RTS application requires a live video stream for user-based targeting. Receiving frames and updating the GUI is commonly offloaded to a background thread to ensure smooth user interaction. With our framework, separate communication clients can be used for controlling the RTS and for receiving streaming data, such as angles, distances and images.

**Data Flow and Memory Layout** Figure B.6 shows the memory layout of the implemented RTS simulator. Control parameters, system states and measurements are stored in separate register blocks. Access to individual blocks can be synchronized to ensure data consistency. This leads to a well-structured system architecture, simplifies user-level data synchronization and allows easier decoupling of the simulation loop, data registers and client requests.

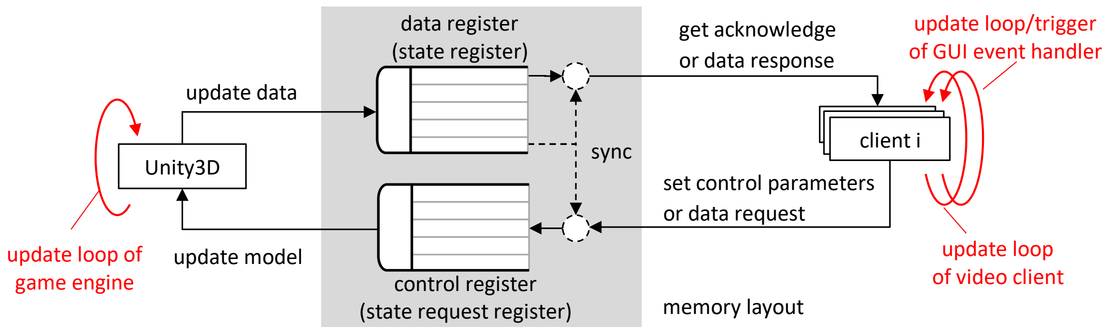


Figure B.6: Data flow diagram (black) and memory layout (gray) of the RTS simulator. The system contains multiple processing loops (red).

### B.4.2 Converting Coordinate System Handedness

Unity3D uses a left-handed coordinate system [105], while most RTS models use right-handed coordinate systems [9, 106]. The kinematic models proposed in chapter 3 are converted from right-handed to left-handed coordinates systems in chapter 4 for simulation with Unity3D.

However, no implementation details are provided for the particular conversion.

Note that coordinate system handing of an arbitrary system can be converted by inverting any axis, whereas this particular setup requires dedicated conversions.

**Mesh Vertices** In this work, we converted left-handed and right-handed coordinate systems by inverting the $x$-coordinate of 3D points, using

$$\tilde{\mathbf{x}}_r = \mathbf{S}_x \tilde{\mathbf{x}}_l \qquad \mathbf{S}_x = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{S}_x^{-1} = \mathbf{S}_x \qquad \tilde{\mathbf{x}}_l = \mathbf{S}_x \tilde{\mathbf{x}}_r \tag{B.1}$$

where $\tilde{\mathbf{x}}_r$ and $\tilde{\mathbf{x}}_l$ are corresponding 3D points in the left-handed and right-handed coordinate system, and $\mathbf{S}_x$ is a *signature matrix*, which inverts the x coordinate of a 3D point.

**Camera and Image Coordinate Systems** The camera projection matrix given in Eqn. 3.29 defines the perspective projection matrix of a right-handed coordinate system according to the pinhole camera model as described in the book by Hartley and Zisserman [44]. While the default camera parameters of Unity3D are not sufficient for this camera model, Unity3D provides access to the transformation matrices used for rendering. By defining the matrices manually, camera models like the pinhole model can be simulated.

In this work, we integrated all required vertex transformations into the default vertex and object transformation pipeline of Unity3D and OpenGL. This has certain advantages, such as not disturbing built-in Unity3D methods, reusing existing GPU Shader programs without modifications, and an implementation which is less prone to errors, since the default behavior of the 3D engine remain unaffected. This also ensures that the engine documentation can be used without modifications, which we believe is particularly helpful to reduce the risk of implementation errors.

The default transformation pipeline of OpenGL is shown in Fig. B.7. Model, world and camera spaces are defined by Unity3D, the related OpenGL matrices for transforming vertices are updated automatically during rendering. The reader is referred to the Unity3D user manual [105] for details about Unity3D matrices and transformations.

Similarly, we adapted the handing of the projection matrix $\mathbf{P}$ given in Eqn. 3.29 to match the Unity3D specifications. Note that a signature matrix is an *involutory matrix*, hence it is its own inverse.

Vertex data must be converted between the camera space of Unity3D and OpenGL, if the default OpenGL pipeline should be used. Once converted, OpenGL performs projection, clipping and rasterization. The result is then rendered to the screen or read back to Unity3D. Details about the rendering and transformation pipelines can be found in the book by Seller et al. [96].

In Unity3D, the *Texture2D* class defines a 2D image container. The allocated memory

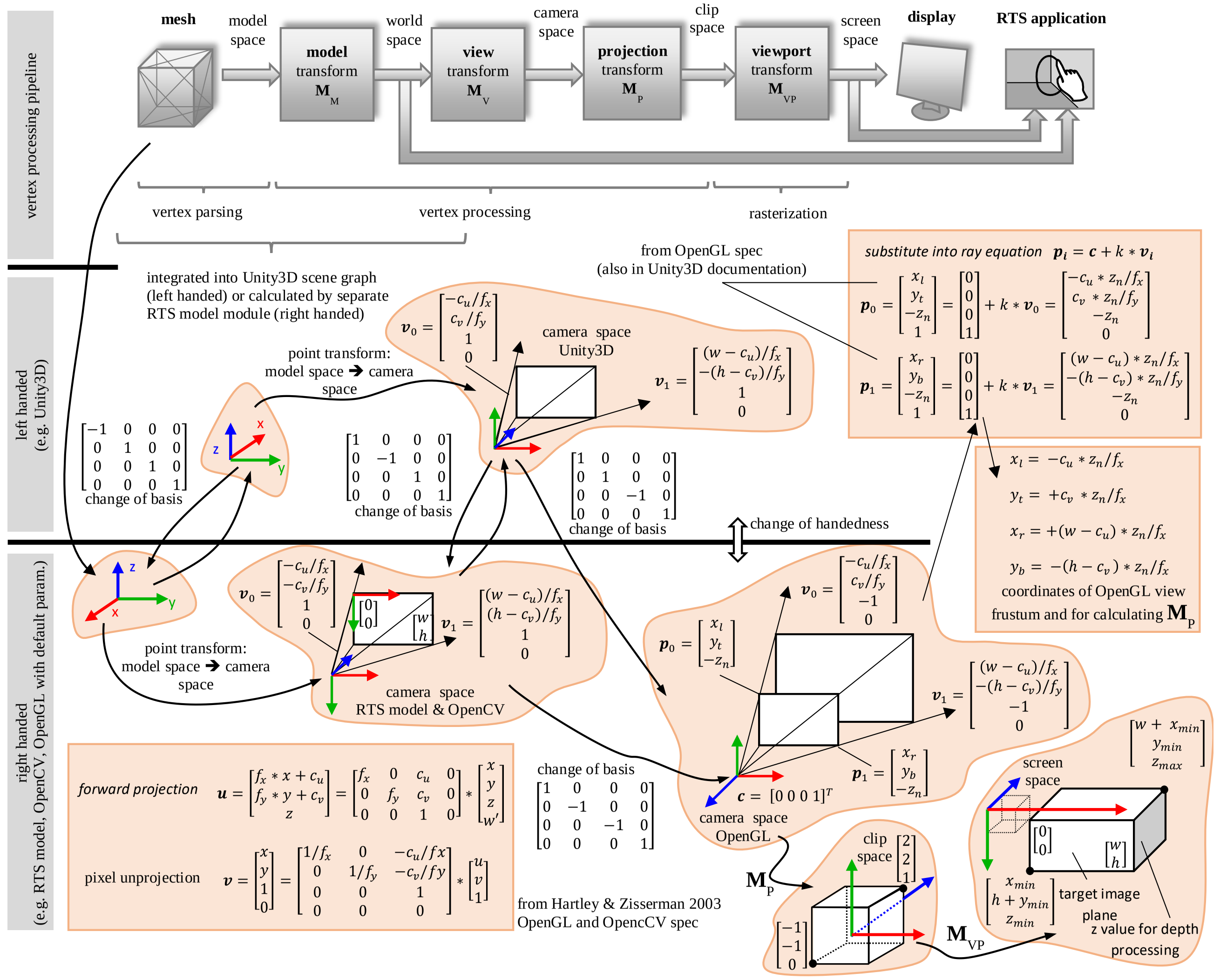

vertex processing pipeline
mesh
model space
model transform $\mathbf{M}_M$
world space
view transform $\mathbf{M}_V$
camera space
projection transform $\mathbf{M}_P$
clip space
viewport transform $\mathbf{M}_{VP}$
screen space
display
RTS application
vertex parsing
vertex processing
rasterization
left handed (e.g. Unity3D)
right handed (e.g. RTS model, OpenCV, OpenGL with default param.)
integrated into Unity3D scene graph (left handed) or calculated by separate RTS model module (right handed)
change of basis
point transform: model space ➔ camera space
camera space Unity3D
camera space RTS model & OpenCV
camera space OpenGL
change of handedness
from OpenGL spec (also in Unity3D documentation)
substitute into ray equation $p_i = c + k * v_i$
$x_l = -c_u * z_n/f_x$
$y_t = +c_v * z_n/f_x$
$x_r = +(w - c_u) * z_n/f_x$
$y_b = -(h - c_v) * z_n/f_x$
coordinates of OpenGL view frustum and for calculating $\mathbf{M}_P$
forward projection
pixel unprojection
from Hartley & Zisserman 2003 OpenGL and OpencCV spec
$c = [0\ 0\ 0\ 1]^T$
clip space
screen space
target image plane
z value for depth processing
$\mathbf{M}_P$
$\mathbf{M}_{VP}$

Figure B.7: Vertex transformation and rendering pipeline according to the OpenGL specification [95] (top). Converting the RTS camera frame between the RTS model (right-handed), Unity3D (left-handed) and OpenGL (default properties). The RTS camera model is conform with the book of Hartley and Zisserman [44] and the OpenCV library [19]. We convert the camera space of the RTS model to the camera space of Unity3D for convenient usage of default scripts without modifications.

block is located on the random-access memory (RAM) and directly accessible by the GPU. The *RenderTexture* class defines a GPU buffer object, which can be used as rendering target[4]. The *Camera* class defines a rendering node and represents both, a 3D node within the scene graph and a rendering definition, which can include one or more rendering passes.

Figure B.7 shows the alignment between the derived model, Unity3D and OpenGL. The kinematic model is converted back and forth between left and right-handed representation when applying model control parameters or reading out the actual system state. The conversion can be simply described as change of basis. While Figure B.7 contains all required conversions for mesh vertex and RTS model transformations, the camera matrix of the system must be incorporated into the rendering pipeline of the graphics engine. By doing so, the default behavior of Unity3D, in particular of existing behavior scripts and GPU Shaders can be preserved.

Let $\mathbf{M}_P$ be the OpenGL projection matrix according to Figure B.7. Then, we can modify the projection matrix $\mathbf{M}_P$ so that it renders the same image as the OpenCV pinhole camera model, except for rasterization and clipping effects.
The default OpenGL projective transform $\mathbf{M}_P$ in column-major order is given by

$$\mathbf{M}_P = \begin{bmatrix} \frac{2z_n}{x_r-x_l} & 0 & \frac{x_r+x_l}{x_r-x_l} & 0 \\ 0 & \frac{2z_n}{y_t-y_b} & \frac{y_t+y_b}{y_t-y_b} & 0 \\ 0 & 0 & -\frac{z_f+z_n}{z_f-z_n} & -\frac{2z_n z_f}{z_f-z_n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{B.2}$$

where the six parameters $x_r$, $x_l$, $y_t$, $y_b$, $z_n$ and $z_f$ defines the distances from the origin of the camera space to the right, left, top, bottom, near and far plane of the view frustum, respectively[5]. $\mathbf{M}_P$ is used for transforming 3D points from the camera space into the clipping space and also for converting the clipped 3D points into normalized device coordinates (NDC).

The four parameters ($x_r$, $x_l$, $y_t$, $y_b$) can be derived by reprojecting pixels from the OpenCV pinhole camera model into the camera space, changing the basis of the camera frame to the OpenGL camera coordinate system, and then calculating the near and far points of the view frustum. Then, the default OpenGL projective transform $\mathbf{M}_P$ in column-major order is fully defined by the camera intrinsics, by Eqn. B.2 and by following four frustum points:

[4] A *RenderTexture* and which provides direct access to a *frame buffer object*. See [105] and [95] for details.

[5] For perspective cameras, the distances for right, left, top and bottom plane are measured along the near clipping plane.

$$\left.\begin{aligned} x_r &= -c_x \frac{z_n}{f_x} & x_l &= (w - c_x)\frac{z_n}{f_x} \\ y_t &= c_y \frac{z_n}{f_y} & y_b &= (c_y - h)\frac{z_n}{f_y} \end{aligned}\right\} \tag{B.3}$$

**Video Streaming** The final result should be rendered to a GPU texture buffer which matches the image resolution $(w, h)$. By using the texture buffer as *render target* of a *Camera* node, Unity3D updates the viewport matrix $\mathbf{M}_{VP}$ for the related rendering passes to

$$\mathbf{M}_{VP} = \begin{bmatrix} \frac{w}{2} & 0 & 0 & x_{min} + \frac{w}{2} \\ 0 & -\frac{h}{2} & 0 & y_{min} + \frac{h}{2} \\ 0 & 0 & z_{max} - z_{min} & z_{min} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.4}$$

where $\begin{bmatrix} x_{min} & y_{max} \end{bmatrix}^T$ is the pixel offset in the target texture, and $z_{min}$ and $z_{max}$ are used for depth processing.

The rendered image can be transferred from the GPU texture buffer to the GPU texture buffer using the API method *ReadPixels* for further processing and video streaming.

**Image Effects** Nonlinear image effects, such as lens distortions, vignetting and exposure, can be simulated using GPU Shaders. If applied as post-processing step to the standard rendering pipeline, the Unity3D API function *Graphics.Blit* can be used.

## B.5 Discussion

In this chapter, we complemented the description of our prototyping and simulation framework by software and implementation details. In particular, we completed the in chapter 3 and chapter 4 proposed framework for generating a efficient ecosystem for interactive RTS algorithm development.

**Integration of Existing Drivers** We discussed how existing drivers can be integrated into a software ecosystem for heterogeneous systems, how gRPC can be used for cross-platform IPC, and how code generators for ready-to-use servers and clients can be extracted from existing code examples.

**Model Conversations** Furthermore, we discussed the required matrix conversions for combining and simulating models with different handedness. The proposed integration of the OpenCV camera model into the existing Unity3D and OpenGL rendering pipeline proved to be particular useful for simulating calibrated cameras. By using our method, the default functionality of the rendering pipeline is largely preserved, which allows using most existing Unity3D and OpenGL scripts without modification. We think that this is particular important to keep the effort for developing and maintaining RTS simulators reasonable low.

**Game and Rendering Engines for Device Simulation** The proposed setup has proven to be useful not only for simulations of geometric, kinematic and dynamic models, but also for running user tests with mockups during early software design phases. Scene graph models of game engines are flexible and efficient, yet well structured and easy to understand. As rendering engines are an integral part of game engines, and they are particularly optimized for high quality graphics and real-time rendering, the simulation of image sensors with low latency and high update rates can easily be achieved. While not discussed in great detail in this work, another benefit is the simulation of camera effects by means of GPU Shaders, such as lens distortions or vignetting.

**Simulator Accuracy** The simulator accuracy is mainly limited by the geometric model accuracy and the numerical precision of the graphics engine Unity3D. While the miss-match between geometric model and real device can be reduced by numerical optimization as proposed in Sec. 4.6, Unity3D is based on 32 bit floating point precision which can not be altered [105]. Advanced techniques could be applied, such as scaling the nominal world units, splitting and scaling object space or image space regions into multiple parts or implementing methods like ray casting explicitly using higher precision arithmetic.

**Latency and Complexity** The server-client architecture of the framework increases the latency and complexity by adding additional third party library dependencies as well as an additional software layer to the communication stack. This might be acceptable for research and prototyping, but not for production code. For industry-oriented research, we suggest to avoid large deviations from the target ecosystem, like adding an additional communication layer as proposed in this work. However, the discussed device simulation can also be applied to the designated programming language of the instrument SDK. Hereby, a mock of the instrument driver can be extracted instead of using gRPC on top of the SDK, whereas the majority of the proposed implementation method can applied as-is.

To summarize, we experienced a great advantage in using game engines for device simulation. Simulated mockups helped us with finding suitable user interfaces and discarding impractical workflows in early design phases. We think that the benefits of custom simulation systems outweighed the effort for creating and maintaining these systems, especially for exploratory research, but also for software design, test and verification in industry.

# C List of Acronyms

| | |
|---|---|
| **API** | application programming interface |
| **AR** | Augmented Reality |
| **BIM** | building information modeling |
| **CAD** | comuter-aided design |
| **CLR** | Common Language Runtime |
| **DH** | Denavit-Hartenberg |
| **DOF** | degree of freedom |
| **EDM** | electronic distance meter |
| **EM** | Expectation Maximization |
| **GoF** | Gang of Four |
| **GPU** | grapics processing unit |
| **GUI** | graphical user interface |
| **GUM** | Guide to the Expression of Uncertainty |
| **HIL** | hardware-in-the-loop |
| **IBM** | image-based measurement |
| **ICP** | Iterative Closest Point |
| **ILP** | linear integer programming |
| **IP** | interest point |
| **IPC** | inter-process communication |
| **LIDAR** | light detection and ranging |
| **MAD** | median absolute deviation |
| **MCS** | Monte-Carlo Simulation |
| **NDC** | normalized device coordinates |
| **PROTOBUF** | Google Protocol Buffer |
| **RAM** | random-access memory |
| **RANSAC** | Random Sample Consesus |
| **ROI** | region of interest |
| **ROS** | Robot Operation System |
| **RPC** | remote procedure call |
| **RTS** | robotic total station |
| **SDK** | software development kit |
| **SLAM** | simultaneous localization and mapping |
| **SQP** | sequential quadratic programming |
| **SVD** | Singular Value Decomposition |
| **SWIG** | Simplified Wrapper and Interface Generator |
| **TSP** | traveling salesman problem |
| **UAV** | unmanned aerial vehicle |
| **UI** | user interface |

# D

# About the Author

Christoph Hubert Klug received the M.S. degree in Telematics from the Technical University of Graz (2012). After suspending his university career to work in the Automotive industry as an embedded software designer for several years, he is currently working as a research assistant at the Virtual Reality and Visualisation Research Centre GmbH (VRVis) in cooperation with the Institute of Computer Graphics and Vision at the Technical University of Graz (ICG TU Graz), where he is finishing his Ph.D. degree in computer sciences. He is currently putting all his energy in enhancing workflows and algorithms for robotic total stations using Computer Vision methods with special regard to usability and accuracy. When not tinkering with robotic total stations, Christoph is totally absorbed not only by everything about computer vision, computer graphics and augmented reality. His recent research interests also covers geometric modeling, human-computer interaction (HCI) and machine learning. In his free time he likes to travel the world with his beloved wife Karina, to do all kind of sports and to teach his cat funny tricks.

# Bibliography

[1] Aho, A., Lam, M., Sethi, R., and Ullman, J. (2007). *Compilers: Principles, Techniques, & Tools*. Pearson/Addison-Wesley, Boston, USA. (page 137)

[2] Alexandrescu, A., Stan, A., Botezatu, N. A., and Caraiman, S. (2016). Real-time interprocess communication in heterogeneous programming environments. In *Proceedings of the International Conference on System Theory, Control and Computing*, pages 283–288. (page 15)

[3] Amann, M.-C., Bosch, T. M., Lescure, M., Myllylae, R. A., and Rioux, M. (2001). Laser ranging: A critical review of usual techniques for distance measurement. *Optical Engineering*, 40(1):10–19. (page 2)

[4] Andaluz, V. H., Canseco, P. A., Rosales, A., Roberti, F., and Carelli, R. (2012). Multilayer scheme for the adaptive cooperative coordinated control of mobile manipulators. In *Proceedings of the IEEE International Conference on Industrial Electronics, Control, and Instrumentation*, pages 2737–2743. (page 15)

[5] Andaluz, V. H., Chicaiza, F. A., Gallardo, C., Quevedo, W. X., Varela, J., Sánchez, J. S., and Arteaga, O. (2016). Unity3D-MATLAB simulator in real time for robotics applications. In *Proceedings of the International Conference on Augmented Reality, Virtual Reality, and Computer Graphics*, Lecture Notes in Computer Science Book Series, pages 246–263, Cham, Switzerland. Springer International Publishing. (page 15)

[6] Andaluz, V. H., Ortiz, J. S., Pérez, M., Roberti, F., and Carelli, R. (2014). Adaptive cooperative control of multi-mobile manipulators. In *Proceedings of the IEEE International Conference on Industrial Electronics, Control, and Instrumentation*, pages 2669–2675. (page 15)

[7] Aristidou, A., Lasenby, J., Chrysanthou, Y., and Shamir, A. (2018). Inverse kinematics techniques in computer graphics: A survey. *Computer Graphics Forum*, 37:35–58. (page 38, 39)

[8] Arun, K. S., Huang, T. S., and Blostein, S. D. (1987). Least-squares fitting of two 3-D point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(5):698–700. (page 126, 127)

[9] Awange, J. and Grafarend, E. W. (2005). *Solving Algebraic Computational Problems in Geodesy and Geoinformatics: The Answer to Modern Challenges*. Springer Berlin Heidelberg, Germany. (page 55, 140)

[10] Barker, L. K. (1983). Vector-algebra approach to extract Denavit-Hartenberg parameters of assembled robot arms. *NASA Technical Paper 2191*. (page 7, 14, 28, 37)

[11] Barlow, R. (1989). *Statistics: A Guide to the use of Statistical Methods in the Physical Sciences*. Wiley, Chichester, England New York. (page 60)

[12] Bartsch, H.-J. (2015). *Handbook of Mathematical Formulas*. Elsevier Science. (page 123)

[13] Beazley, D. M. and Others (1996). SWIG: An easy to use tool for integrating scripting languages with c and c++. In *Proceedings of the Tcl/Tk Workshop*, volume 4. USENIX Association. (page 136)

[14] Besl, P. J. and McKay, H. D. (1992). A method for registration of 3–D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256. (page 17)

[15] Betts, M., Robinson, G., Burton, C., Leonard, J., Sharda, A., and Whittington, T. (2015). Global construction 2030: A global forecast for the construction industry to 2030. (page 120, 121)

[16] Bosché, F. (2010). Automated recognition of 3D cad model objects in laser scans and calculation of as-built dimensions for dimensional compliance control in construction. *Advanced Engineering Informatics*, 24:107–118. (page 5)

[17] Bosché, F. (2012). Plane-based registration of construction laser scans with 3D/4D building models. *Advanced Engineering Informatics*, 26:90–102. (page 5)

[18] Box, G. E. P. and Muller, M. E. (1958). A note on the generation of random normal deviates. *The Annals of Mathematical Statistics*, 29(2):610–611. (page 55, 80)

[19] Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*. (page 129, 142)

[20] Carpin, S., Lewis, M., Wang, J., Balakirsky, S., and Scrapper, C. (2007). USARSim: A robot simulator for research and education. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1400–1405. IEEE. (page 14)

[21] Chen, Y. and Medioni, G. (1992). Object modelling by registration of multiple range images. *Image and Vision Computing*, 10(3):145–155. (page 17)

[22] Chittawadigi, R. G., Hayat, A. A., and Saha, S. K. (2013). Geometric model identification of a serial robot. In *Proceedings of the IFToMM International Symposium on Robotics and Mechatronics*, volume 3, Lorong Bakar Batu, Singapore. Research Publishing Services. (page 14, 23, 24, 37)

[23] Coaker, L. H. (2009). Reflectorless total station measurements and their accuracy, precision and reliability. BSc. Thesis, University of Southern Queensland, Faculty of Engineering and Surveying. (page 13, 73)

[24] Connolly, C. (2009). Technology and aapplications of ABB RobotStudio. *Industrial Robot: The International Journal of Robotics Research and Applications*, 36(6):540–545. (page 14)

[25] Corporation, T. (2011). Imaging station is series, instruction manual. (page 13)

[26] Crossley, F. R. E. (1967). The permutations of kinematic chains of eight members or less from the graph theoretic viewpoint. In *Developments in Theoretical and Applied Mechanics*, volume 2, pages 467–486, Oxford, UK. Pergamon Press. (page 14)

[27] Delp, S. L., Anderson, F. C., Arnold, A. S., Loan, P., Habib, A., John, C. T., Guendelman, E., and Thelen, D. G. (2007). OpenSim: Open-source software to create and analyze dynamic simulations of movement. *IEEE Transactions on Biomedical Engineering*, 54(11):1940–1950. (page 14)

[28] Denavit, J. and Hartenberg, R. S. (1955). A kinematic notation for lower-pair mechanisms based on matrices. *Journal of Applied Mechanics*, 22:215–221. (page 7, 14, 20)

[29] Eggert, D., Lorusso, A., and Fisher, R. (1997). Estimating 3-D rigid body transformations: A comparison of four major algorithms. *Machine Vision and Applications*, 9(5-6):272–290. (page 126)

[30] Ehrgott, M. (2005). *Multicriteria Optimization*. Lecture Notes in Economics and Mathematical Systems. Springer. (page 49)

[31] Ehrhart, M. (2017). *Applications of Image-Assisted Total Stations: Concepts, Experiments, Results and Calibration*. PhD Thesis, Graz University of Technology, Institute of Engineering Geodesy and Measurement Systems, Graz. (page 3, 6, 16, 18, 20, 38, 57)

[32] Ehrhart, M. and Lienhart, W. (2015). Image-based dynamic deformation monitoring of civil engineering structures from long ranges. In *Proceedings of the Conference of the Society of Photo-Optical Instrumentation Engineers*, volume 9405. SPIE. (page 16, 128)

[33] El-Sherbiny, A., Elhosseini, M. A., and Haikal, A. Y. (2018). A comparative study of soft computing methods to solve inverse kinematics problem. *Ain Shams Engineering Journal*, 9:2535 – 2548. (page 38, 39)

[34] Everett, L., Driels, M., and Mooring, B. (1987). Kinematic modelling for robot calibration. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 4, pages 183–189. IEEE. (page 39)

[35] Faro Technologies (2019). `https://www.faro.com`. [Online; Accessed 03 February 2019]. (page 121)

[36] Fathi, H. and Brilakis, I. (2013). A videogrammetric as-built data collection method for digital fabrication of sheet metal roof panels. *Advanced Engineering Informatics*, 27(4):466–476. (page 16)

[37] Fischler, M. a. and Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applicatlons to image analysis and automated cartography. *Communications of the ACM*, 24(6):381 – 395. (page 75, 102)

[38] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. (page 133, 135)

[39] Global Construction Perspectives Limited (2019). `https://gcp.global`. [Online; Accessed 04 February 2019]. (page 120)

[40] González-Palacios, M. A., Ortega-Alvarez, C. J., Sandoval-Castillo, J. G., Cuevas-Ledesma, S. M., and Mendoza-patiño, F. J. (2016). The generalized architecture of the spherical serial manipulator. *Advances in Robotics & Automation*, 5(2). (page 38, 39, 40)

[41] Google Inc. (2017). gRPC open-source universal RPC framework. `http://www.grpc.io`. [Online; Accessed 28 July 2017]. (page 54, 131, 133, 135, 136, 137, 138)

[42] Harrisberger, L. (1965). A number synthesis survey of three-dimensional mechanisms. *Journal of Engineering for Industry*, 87(2):213–218. (page 14)

[43] Hartenberg, R. S. and Denavit, J. (1964). *Kinematic Synthesis of Linkages*. McGraw-Hill Series in Mechanical Engineering. McGraw-Hill. (page 14)

[44] Hartley, R. and Zisserman, A. (2003). *Multiple View Geometry in Computer Vision*. Cambridge University Press, Cambridge, UK, second edition. (page 18, 36, 37, 76, 123, 125, 126, 128, 129, 141, 142)

[45] Hayat, A. A., Chittawadigi, R. G., Udai, A. D., and Saha, S. K. (2013). Identification of Denavit-Hartenberg parameters of an industrial robot. In *Proceedings of the Conference on Advances In Robotics*, pages 55:1–55:6, New York, USA. ACM. (page 14, 23, 24, 37)

[46] Horn, B. K. P. (1987). Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642. (page 102, 126)

[47] Horn, B. K. P., Hilden, H. M., and Negahdaripour, S. (1988). Closed-form solutions of absolute orientation using orthonormal matrices. *Journal of the Optical Society of America A*, 5:1127–1135. (page 100, 126)

[48] Hough, P. V. C. (1962). Method and means for recognizing complex patterns. US Patent 3069654. (page 17)

[49] Hu, Y. and Meng, W. (2016). ROSUnitySim: Development and experimentation of a real-time simulator for multi-unmanned aerial vehicle local planning. *Simulation*, 92(10):931–944. (page 15, 137)

[50] Hulik, R., Spanel, M., Smrz, P., and Materna, Z. (2014). Continuous plane detection in point-cloud data based on 3d hough transform. *Journal of Visual Communication and Image Representation*, 25(1):86–97. (page 17)

[51] IEEE Std 754-2008 (2008). IEEE standard for floating-point arithmetic (revision of IEEE std 754-1985). Standard, IEEE Computer Society. (page 52, 56, 60)

[52] ISO 17123-3:2001 (2001). Optics and optical instruments – field procedures for testing geodetic and surveying instruments. Standard, International Organization for Standardization, Geneva, Switzerland. (page 86)

[53] ISO 19650-1:2018 (2018). Organization and digitization of information about buildings and civil engineering works, including building information modelling (BIM) – information management using building information modelling – part 1: Concepts and principles. Standard, International Organization for Standardization, Geneva, Switzerland. (page 121)

[54] ISO 19650-2:2018 (2018). Organization and digitization of information about buildings and civil engineering works, including building information modelling (BIM) – information management using building information modelling – part 2: Delivery phase of the assets. Standard, International Organization for Standardization, Geneva, Switzerland. (page 121)

[55] ISO/IEC 23271:2012 (2012). Information technology – common language infrastructure (CLI). Standard, International Organization for Standardization, Geneva, Switzerland. (page 133)

[56] Jadidi, H., Ravanshadnia, M., Hosseinalipour, M., and Rahmani, F. (2015). A step-by-step construction site photography procedure to enhance the efficiency of as-built data visualization: A case study. *Visualization in Engineering*, 3(1):1–12. (page 16)

[57] JCGM 100:2008 (2008). Evaluation of measurement data – guide to the expression of uncertainty in measurement (GUM). Standard, International Organization for Standardization, Geneva, Switzerland. (page 56, 59, 61, 79, 80, 111, 114)

[58] Joubair, A. and Bonev, I. A. (2015). Kinematic calibration of a six-axis serial robot using distance and sphere constraints. *International Journal of Advanced Manufacturing Technology*. (page 39)

[59] Julian, K., Ganapathi Subramanian, A., Lee, J., and Faghihi, V. (2012). Robotic total station and BIM for quality control. In *eWork and eBusiness in Architecture,*

*Engineering and Construction: Proceedings of the European Conference on Product and Process Modeling in the Building and Construction Industry*, volume 9, pages 717–722, Reykjavik, Iceland. CRC Press. (page 121)

[60] Juretzko, M. (2004). *Reflektorlose Video-Tachymetrie - Ein Integrales Verfahren zur Erfassung Geometrischer und Visueller Informationen.* PhD Thesis, Ruhr University Bochum, Faculty of Civil Engineering. (page 17, 69, 70, 73, 109)

[61] Kabsch, W. (1976). A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A*, 32(5):922–923. (page 127)

[62] Klasing, K., Althoff, D., Wollherr, D., and Buss, M. (2009). Comparison of surface normal estimation methods for range sensing applications. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3206–3211. (page 76, 77)

[63] Klug, C., Arth, C., Schmalstieg, D., and Gloor, T. (2018a). Measurement uncertainty analysis of a robotic total station simulation. In *Proceedings of the IEEE International Conference on Industrial Electronics, Control, and Instrumentation*, pages 2576–2582. (page 7, 10, 19)

[64] Klug, C., Arth, C., Schmalstieg, D., and Gloor, T. (2018b). Semi-automatic registration of a robotic total station and a cad model without control points. In *Proceedings of the IEEE International Conference on Industrial Electronics, Control, and Instrumentation*, pages 2631–2638. (page 7, 11, 100)

[65] Klug, C., Schmalstieg, D., and Arth, C. (2017). Measuring human-made corner structures with a robotic total station using support points, lines and planes. In *Proceedings of the International Conference on Computer Vision Theory and Applications*, volume 6, pages 17–27. INSTICC, SciTePress. (page 8, 45, 69, 71, 87, 88, 91, 92, 96, 97, 109, 114)

[66] Klug, C., Schmalstieg, D., Gloor, T., and Arth, C. (2018c). A complete workflow for automatic forward kinematics model extraction of robotic total stations using the denavit-hartenberg convention. *Journal of Intelligent and Robotic Systems.* (page 7, 9, 19, 20, 37, 129)

[67] Klug, C., Schmalstieg, D., Gloor, T., and Arth, C. (2019). On using 3D support geometries for measuring human-made corner structures with a robotic total station. In *Computer Vision, Imaging and Computer Graphics – Theory and Applications (Revised Selected Papers of VISIGRAPP 2017)*, Communications in Computer and Information Science, pages 352–374. Springer International Publishing. (page 7, 12, 69, 71)

[68] Koenig, N. and Howard, A. (2004). Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2149–2154. IEEE. (page 14)

[69] Kucuk, S. and Bingul, Z. (2006). Robot kinematics: Forward and inverse kinematics. In *Industrial Robotics: Theory, Modelling and Control*. pro literatur Verlag (plV) Robert Mayer-Scholz, Mammendorf, Germany. (page 21)

[70] Lachat, E., Landes, T., and Grussenmeyer, P. (2017). Investigation of a combined surveying and scanning device: The trimble sx10 scanning total station. *Sensors*, 17(4). (page 4)

[71] Leys, C., Ley, C., Klein, O., Bernard, P., and Licata, L. (2013). Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology*, 49(4):764–766. (page 89, 114)

[72] MathWorks Inc. (2017). MATLAB Optimization Toolbox. `https://mathworks.com/products/optimization.html`. [Online; Accessed 28 July 2017]. (page 49)

[73] Mattingly, W. A., j. Chang, D., Paris, R., Smith, N., Blevins, J., and Ouyang, M. (2012). Robot design using Unity for computer games and robotic simulations. In *Proceedings of the IEEE International Conference on Computer Games*, pages 56–59. IEEE. (page 14, 15, 137)

[74] McCarthy, J. M. and Soh, G. S. (2011). *Geometric Design of Linkages*. Interdisciplinary Applied Mathematics. Springer New York, second edition. (page 14)

[75] Megahed, S. (2012). Inverse kinematics of spherical wrist robot arms: Analysis and simulation. *Journal of Intelligent and Robotic Systems*, 5(3):211–227. (page 39)

[76] Mehdi, H. M. (2008). *New Algorithms in Rigid-Body Registration and Estimation of Registration Accuracy*. PhD Thesis, Queen's University, Department of Electrical and Computer Engineering, Kingston, Canada. (page 17, 111)

[77] Meng, W., Hu, Y., Lin, J., Lin, F., and Teo, R. (2015). ROS+Unity: An efficient high-fidelity 3D multi-UAV navigation and control simulator in GPS-denied environments. In *Proceedings of the IEEE International Conference on Industrial Electronics, Control, and Instrumentation*, pages 2562–2567. IEEE. (page 15, 137)

[78] Microsoft Corporation (2019). .NET Framework. `https://dotnet.microsoft.com/`. [Online; Accessed 10 February 2019]. (page 133)

[79] Nguyen, T., Grasset, R., Schmalstieg, D., and Reitmayr, G. (2013). Interactive syntactic modeling with a single-point laser range finder and camera. *Proceedings of the IEEE International Symposium on Mixed and Augmented Reality*, pages 107–116. (page 17, 102, 106)

[80] Nocedal, J. and Wright, S. (2006). *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York. (page 49)

[81] Oxford Economics Limited (2019). `https://www.oxfordeconomics.com`. [Online; Accessed 04 February 2019]. (page 120)

[82] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). ROS: an open-source robot operating system. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 3. Kobe, Japan. (page 132)

[83] Rajeevlochana, C. G. and Saha, S. K. (2011). RoboAnalyzer: 3D model based robotic learning software. In *Proceedings of the International Conference on Multibody Dynamics*, volume 6, pages 3–13, Vijayawada, India. (page 14, 20)

[84] Rajeevlochana, C. G., Saha, S. K., and Kumar, S. (2012). Automatic extraction of DH parameters of serial manipulators using line geometry. In *Proceedings of the Joint International Conference on Multibody System Dynamics*, volume 2, Stuttgart, Germany. (page 7, 14, 28, 37)

[85] Reese, G. (2000). *Database programming with JDBC and Java*. O'Reilly, Sebastopol, CA, USA, second edition. (page 133)

[86] Reuleaux, F. (1876). *The Kinematics of Machinery: Outlines of a Theory of Machines*. Macmillan and CO. (page 14)

[87] Rohmer, E., Singh, S. P. N., and Freese, M. (2013). V-REP: A versatile and scalable robot simulation framework. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326. IEEE. (page 14)

[88] Saha, S. K. (2014). *Introduction to Robotics*. McGraw Hill, second edition. (page 20, 37)

[89] Scherer, M. (2002). Advantages of the integration of image processing and direct coordinate measurement for architectural surveying - development of the system TOTAL. In *Proceedings of the International Congress of the International Federation of Surveyors*. (page 16)

[90] Scherer, M. (2004). Intelligent scanning with robot-tacheometer and image processing: A low cost alternative to 3d laser scanning? In *Proceedings of the Working Week of the International Federation of Surveyors*, pages 22–27. (page 73)

[91] Scherer, M. and Lerma, J. L. (2009). From the conventional total station to the prospective image assisted photogrammetric scanning total station: Comprehensive review. *Journal of Surveying Engineering*, 135(4):173–178. (page 16)

[92] Schmalstieg, D. and Höllerer, T. (2016). *Augmented Reality: Principles and Practice*. Addison-Wesley Professional, Boston, USA. (page 6)

[93] Schnabel, R., Wahl, R., and Klein, R. (2007). Efficient RANSAC for point-cloud shape detection. *Computer Graphics Forum*, 26(2):214–226. (page 17)

[94] Schneider, P. and Eberly, D. (2003). *Geometric Tools for Computer Graphics*. Boston Morgan Kaufmann Publishers, Amsterdam. (page 76, 78, 123, 124)

[95] Segal, M. and Akeley, K. (2017). The OpenGL graphics system: A specification. version 4.5 (core profile). `https://www.khronos.org/registry/OpenGL/specs/gl/glspec45.core.pdf`. [Online; Accessed 28 July 2017]. (page 57, 58, 142, 143)

[96] Sellers, G., Wright, R. S., and Haemel, N. (2015). *OpenGL Superbible: Comprehensive Tutorial and Reference*. OpenGL Series. Addison-Wesley, seventh edition. (page 141)

[97] Sheth, P. N. and Uicker, J. J. (1971). A generalized symbolic notation for mechanisms. *Journal of Engineering for Industry*, 93(1):102–112. (page 14)

[98] Siciliano, B. and Khatib, O. (2016). *Springer Handbook of Robotics*. Springer Handbooks. Springer International Publishing. (page 14)

[99] Siemens PLM Software (2011). Robcad Tecnomatix. `https://www.plm.automation.siemens.com/en/products/tecnomatix/manufacturing-simulation/robotics/robcad.shtmls`. [Online; Accessed 28 July 2017]. (page 14)

[100] Siu, M.-F., Lu, M., and AbouRizk, S. (2013). Combining photogrammetry and robotic total stations to obtain dimensional measurements of temporary facilities in construction field. *Visualization in Engineering*, 1(1). (page 16)

[101] Stone, H. W. (2012). *Kinematic Modeling, Identification, and Control of Robotic Manipulators*. The Springer International Series in Engineering and Computer Science. Springer US. (page 14)

[102] Tellinghuisen, J. (2001). Statistical error propagation. *The Journal of Physical Chemistry A*, 105(15):3917–3921. (page 60)

[103] Tonello, S., Zanetti, G. P., Finotto, M., Bortoletto, R., Tosello, E., and Menegatti, E. (2012). WorkCellSimulator: A 3D simulator for intelligent manufacturing. In *Proceedings of the IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pages 311–322, Germany. Springer Berlin Heidelberg. (page 14)

[104] Trevor, A. J. B., Gedikli, S., Rusu, R. B., and Christensen, H. I. (2013). Efficient organized point cloud segmentation with connected components. *Proceedings of the Workshop on Semantic Perception, Mapping and Exploration*. (page 106)

[105] Unity Technologies (2017). Unity3D: Game engine. `https://unity3d.com`, `https://docs.unity3d.com/Manual`. [Online; Accessed 28 July 2017]. (page 54, 55, 81, 139, 140, 141, 143, 145)

[106] Uren, J. and Price, B. (2010). *Surveying for Engineers*. Palgrave Macmillan. (page 1, 2, 5, 13, 20, 38, 55, 81, 140)

[107] Vanderbei, R. (2008). *Linear Programming: Foundations and Extensions*. International Series in Operations Research & Management Science. Springer New York, New York, USA. (page 110)

[108] Veitschegger, W. K. and Wu, C. H. (1988). Robot calibration and compensation. *IEEE Journal on Robotics and Automation*, 4(6):643–656. (page 14)

[109] Witten, I. H. and Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Inc., second edition. (page 51)

[110] Wu, Y., Klimchik, A., Caro, S., Furet, B., and Pashkevich, A. (2015). Geometric calibration of industrial robots using enhanced partial pose measurements and design of experiments. *Robotics and Computer-Integrated Manufacturing*. (page 39)

[111] Zeiske, K. (2004). Surveying made easy. `https://www.aps.anl.gov/files/APS-Uploads/DET/Detector-Pool/Beamline-Components/Lecia_Optical_Level/Surveying_en.pdf`. [Online; Accessed 22 May 2018]. (page 13, 16)