PATH DETECTION, ESTIMATION AND CORRECTION FOR ROBOT GUIDANCE

Teză destinată obţinerii
titlului ştiinţific de doctor inginer
la
Universitatea Politehnica Timişoara
în domeniul INGINERIE ELECTRONICĂ
ŞI TELECOMUNICAŢII
de către

M.Sc. Markus Herrmann

Conducător ştiinţific:	prof.univ.dr.ing Marius Otesteanu
Referenţi ştiinţifici:	prof.univ.dr. ……..
	prof.univ.dr.ing. …….
	conf.univ.dr.ing. …….

Ziua susţinerii tezei: ……..

Seriile Teze de doctorat ale UPT sunt:

1. Automatică
2. Chimie
3. Energetică
4. Ingineria Chimică
5. Inginerie Civilă
6. Inginerie Electrică
7. Inginerie Electronică şi Telecomunicaţii
8. Inginerie Industrială
9. Inginerie Mecanică
10. Ştiinţa Calculatoarelor
11. Ştiinţa şi Ingineria Materialelor
12. Ingineria sistemelor
13. Inginerie energetică
14. Calculatoare şi tehnologia informaţiei
15. Ingineria materialelor
16. Inginerie şi Management
17. Arhitectură
18. Inginerie civilă și instalații
19. Inginerie electronică, telecomunicații și tehnologii informaționale

Universitatea Politehnica din Timişoara a iniţiat seriile de mai sus în scopul diseminării expertizei, cunoştinţelor şi rezultatelor cercetărilor întreprinse în cadrul şcolii doctorale a universităţii. Seriile conţin, potrivit H.B.Ex.S Nr. 14 / 14.07.2006, tezele de doctorat susţinute în universitate începând cu 1 octombrie 2006.

România, 300159 Timişoara, Bd. Republicii 9,
tel. 0256 403823, fax. 0256 403221
e-mail: editura@edipol.upt.ro

Acknowledgments

Form starting my work on this thesis until writing these final words, it was an exciting journey. Therefore, I would like to thank my supervisor Professor Marius Otesteanu not only for his amazing support, his helpful advises and motivation, but also for giving me the opportunity to go on this journey and believing in me throughout the way.

Furthermore, I would like to thank Professor Franz Quint for all his great help, his extensive feedback and constructive criticism regarding all the issues from the big picture down to nitty-gritty details discussed in the following chapters.

Also, I would like to thank Dr. Werner Neddermeyer, for awaking my interest in robotics and computer vision , for being an inspiration and role model throughout my professional career, for initiating the contact to Professor Otesteanu and for giving me opportunity to start the work on this thesis at VMT Bildverarbeitungssysteme GmbH.

I also owe a big thanks to Dr. Liviu Toma for being such a great companion, keeping my back free whenever it was necessary to focus more on this thesis and for pushing me with incredible enthusiasm towards the finish line.

Last, but not least, I would like to thank my mother, for all the love and confidence she has put in me. With her incredible support, she made it possible for me finishing graduate school and achieving an academic career.

Destinatarii dedicaţiei.

Herrmann, Markus

Path detection, estimation and correction for robot guidance

Teze de doctorat ale UPT, Seria 7, Nr. 85, Editura Politehnica, 200Z, 168 pagini, 39 figuri, 27 tabele.

ISSN:1842-7014

ISBN:978-606-35-0359-7

Cuvinte cheie:
Laser Stripe Sensors, Visual Servoing, Path Correction, Robot Guidance, Feature Detection, Sensor Calibration

Rezumat,

This thesis proposes new methods for robot path correction. These methods are targeting applications of visual beads on wing parts, such as doors, fender, trunk lids, etc. Since such applications require a high demand of accuracy next to quick cycle times, the methods presented are not only targeting the path correction alone, but also improvements of feature detection, calibration, and sensor images. Therefore, this thesis introduces a method for detecting noise points within the images of laser stripe sensor, by applying statistical methods based on Brownian motion with drift. Furthermore, in order to improve feature detection with an acceptable speed, we present a method that combines an Em-ICP with the Douglas Peuker algorithm to achieve much higher execution speed. Finally, the core of this thesis is an algorithm to determine the 6-dimensional correction of the application path of a robot based on sparse stripe scans.

The final topic this thesis addresses is how to make these methods applicable within visual servoing applications. This includes a new approach to motionless calibration for laser stripe sensors and also an adaption of the path correction algorithm in order to provide relative measures.

Contents

List of Figures

Acronyms

BK Name of a vision system with the abbrevation of path correction (german: Bahn Korrektur)

CAD Computer aided drawing

CCD Camera chip type (charge-coupled device)

CLT The central limit theorem

CPUs The central processing unit of a computer

CUDA Programming language for graphic cards by NVidia (Compute Unified Device Architecture)

DH-transformation The denavit hartenberg transformation sheme

DLT Direct linear transformation

DOF Degrees of freedom

DÜRR Name of a company providing paint shop solutions

EcoEMOS Product name of a DÜRR product

EM-Algorithm Algorithm based on estimation maximization principle

EM-ICP Soft assigning ICP algorithm based on the estimation maximization principle

FAW/Volkswagen First Automotive Works, a name of a car manufacturer and joint venture partner of Volkswagen in china

FPGA's Field Programmable Gate Array, a flexible programmable chip

GBM Geometric Brownian motion

GNU "GNU" is not UNIX. A central initiative for open source projects

ICP Iterative closest point, an algorithm for point cloud matching

KUKA A robot manufacturer

LLS Laser line scanner

MAP Maximum a posteriori a stochastic method

OpenMP Multi threading library

PC Personal Computer

QR decomposition Matrix decomposition into an orthogonal matrix and an upper triangular matrix

RPY Roll-Pitch-Yaw, an angle notation

RSI Robot Sensor Interface

RUR Rossum's Universal Robots, a play by Karel Capek

SA-ICP Soft assign ICP algorithm

SCARA Selective Compliance Assembly Robot Arm a type of robot

SVD decomposition Singular value decomposition, a matrix decomposition

TCP Tool center point at the robots wrist

VMT Name of a German vision company

YPR Yaw-Pitch-Roll, an angle notation

Chapter 1

Introduction

Within the recent two centuries, humanity already underwent three industrial revolutions and is standing on the edge of the so-called "fourth industrial" revolution right now. Each of them yielded certain accomplishments, which had major positive impact on the quality of life of many and boosted productivity.

At the beginning of this new industrial revolution, "industry 4.0" the hope is that devices will get smaller and smarter, leading to a multiplicity of smart, elegant helpers which are highly connected and do not in any way remind us of those bulky personal computers we use today. The hope is that technology gets more integrated and more connected with most of the things we commonly use. Multiple smart devices, an internet of things, helping and guiding us within the construction lines, office spaces, and storage.

We also expect that this increased connectivity will leverage the possibility to collect all sorts of data about processes, monitoring quality and spot shortages within production. Everything will be uploaded into the "cloud" and all kinds of complex statistical analyses will be performed within this "big data" to optimize processes and predict business development.And, finally, robots which are already indispensable to production today, are expected to be ubiquitous. In today's vision of the industry 4.0 factory, robots will also appear as a kind of artificial assistant. They are expected to become more collaborative and work next to their human co-workers within the production lines. These collaborative robots do their work without being caged within security fences.

But operating robots next to humans entails not only that robots act more carefully and being more sensitive to their environment in order to not harm their co-workers. It also requires them to handle more complex scenarios, since the work environment they must deal with is the environment of a human, which is usually not specialized for automatic processes. Hence equipping robots with cameras and sensors, making them smarter so they can be more flexible in interacting with their environment will be a crucial demand for the foreseeable future. And yet, reality looks much different. Although there are hundreds of robots and machines on production lines, there are still numerous humans necessary for multiple assembly

steps. Machines are still missing major sensing capabilities and are not yet able to handle situations which are not well-defined. One such not well-defined situation occurs as soon as the parts such robots must deal with are not fixed in position or have irregularities.

This is because machines like robots are built to repeat their work with high accuracy. If a work piece is misplaced, a robot needs to modulate its path. If it just statically repeats the path it once learned, it will fail. To empower a robot to recognize such situations, it needs to be equipped with sensors and software which can determine an exact position of a workpiece and can provide one or multiple corrections to the robot. This type of software is usually applying methods of computer vision and therefore it is called a vision system. These vision systems are using optical devices to detect the parts, their position and irregularities and guide the machines within their application. They exist in multiple variations; some use simple camera technology to observe objects, others apply complex combinations of structured light, laser or other technologies to determine the desired information. Sensor types are usually chosen by the customer demands in accuracy. For some applications, it is sufficient to get a simple one-dimensional measure like distance, while others need highly accurate 6-dimensional [1] measures of the environment where they operate. For some applications, it is sufficient to have a rough global measurement, within the range of a few millimeters, whereas others need very high accuracy, which makes it necessary to take the measurement close to the spot of application to get a higher measurement quality. For high-accuracy applications, it might also be necessary to determine the shape of an object's deviations.

In order to understand the requirements of such systems, one needs to understand the problems arising during production. Further, one also needs to understand where tolerances come from and how they affect the quality of the final results. For instance, as a car progresses through the assembly process, the leeway for tolerances rises. Since every part of the body adds little tolerances to the whole construction, no car will end up in perfect shape. Although these tolerances have been minimized repeatedly during the past decades, they have not completely disappeared yet. On the other hand, the quality of the car strongly depends on keeping those tolerances as low as possible. Therefore, since the goal is to raise the production quality, localizing such sources of tolerance and reducing them is still one of the main problems to solve.

Finally, one can state that there are multiple sources of errors, and they are "adding"[2] up within the different assembly stages. However, this means that keeping the measurements as local as possible for a given process excludes many of these global influences. The path-correction algorithm proposed in this thesis is a good example of such a method. It determines the correction directly at the spot of application, excluding all previous errors within the tolerance chain.

Another problem is that, compared to humans, who can with determine objects within their

[1] Those 6 dimensions are composed of the 3 directions of room space and the corresponding rotations around each room space direction degree of freedom.

[2] The word "adding" should not be interpreted in a mathematical manner, since from a statistical point of view these values do not just simply add up.

visual perception and correlate them within the images of both eyes to a 6 dimensional scenery effortlessly, just by judging their shape and appearances, vision systems do not have such a flexible perception. These systems still need strong features and some kind of geometrical information to be able to determine a model in a 6-dimensional space. A vision system has no human perception; it cannot detect objects as they are but rather concentrates on visually strong characteristic within the object to determine its position.

Another obstacle is that the dimensionality of most sensors is less than the expected dimensionality of the results. An obvious example is a camera, which only provides brightness levels within a 2-dimensional matrix, but which is sometimes expected to measure within 3-dimensional space. Hence one needs to determine a way to extract the missing information about the third dimension from the image. This can be rather complicated, and yet there are multiple approaches to solving this problem today. Each approach has its strength and weaknesses in certain situations. But almost all are based on a mathematical model copying the projection model of the light rays registered by the camera chip. This thesis will provide a basic overview of the common methods and the mathematics behind them in the chapter "State of the Art". Many modern sensors that also includes laser-triangulation sensors are based on mathematical models, which are somewhat similar to the ordinary camera models presented in that chapter.

One sensor type that especially benefits from theses model equations is the laser line sensor. This is a type of structured-light sensor that uses the model equations mentioned above to reconstruct the geometrical characteristics of the observed shape. They often appear as smart sensors, which achieve laser detection in the image data and the geometric reconstruction within CPUs and FPGAs embedded within the device. Nowadays, many robot guidance applications focus on such sensor devices and, therefore, they are the subject of many research projects. Their benefits are obvious, since they provide information about the objects' shape without needing any visual features[3]. In addition, they are also strongly independent of the objects' color or the environmental light. Having such sensors, which can work with surface shape characteristics rather than visual features, opens the door to a whole new range of applications. However, existing algorithms need to be adapted to this kind of measurements, as well. And thus, this whole new set of possibilities also entails a huge demand for research. Hence, this thesis focuses on the two major applications of theses sensors, which are path correction and panel fitting. It introduces state-of-the-art modeling and derives new techniques to apply laser stripe sensors more efficiently. The major focus will be on path correction for sealing bead on wing parts. This thesis will introduce new model equations to determine the correction vectors based on laser stripe sensor data, which are applicable for both path correction and panel fitting. All proposed ideas will be presented as whole concepts, which means that this work not only focuses on the mathematics for determining the correction vectors, but also derives methods to perform calibration and optimize the sensor data, in a way, that fits the requirements of the application cases.

[3]Features that are visible within an image of the object, like an camera image.

1.1 History of assembly line production

The ideas discussed in this thesis are meant to be applied within automated industrial assembly lines for car production. An assembly line breaks the whole production down into multiple steps. This is achieved by dividing the complex process of manufacturing a car into multiple simple steps, which will be performed at numerous so-called workstations.

This concept goes back to Frederick Winslow Taylor, who searched for multiple ways to improve industrial production outcomes. One of his insights among multiple others was that by factoring processes into discrete unambiguous units, the workers will perform significantly better than in doing complex work that consists of multiple steps. His hope was to be able to break down each step into simple, well-documented, and scientifically analyzed processes that can be executed almost automatically. Taylor published his famous ideas, which can be understood as the first fundamentals of horizontal production optimization, in his book *The principals of scientific management* [1]. Shortly after, in 1913, Henry Ford pushed these concepts to the next level by introducing the first assembly production lines in his car production process.

Nevertheless, in those days production lines were full of people and automated workstations were a rarity. Today, more than 100 years later, this image has changed drastically. In multiple areas within many production sites, humans are rarely spotted.

One of the major breakthroughs on the road to perfectly subdivided and mechanical automated production processes, like Taylor imagined them, was the invention of the industrial robot.

1.1.1 Industrial Robots

As already stated in the introduction *[1]*, robots were a major breakthrough for the automated production. Generally spoken, a robot can be seen as a flexible and programmable machinery able to perform complex tasks repetitively and with high accuracy. The idea of robots is quite old and cannot be dated with precision, but the word “robot“, however, originated in the Czech language *(robota meaning "forced labor")* and was introduced in the role play *"Rossum's Univeral Robots (RUR)"* written by Karel Capek [2]. The *"Robots"* within RUR were mechanical creatures meant to replace the human workers and eventually turn on their masters.

Today, robots certainly do replace the work of multiple humans in many cases. Although this might sound scary at first, it is not necessarily so. There are multiple types of robots today and many among them serve irreplaceable tasks. They might be used in places which are dangerous or where humans should not or cannot work. These can be tasks like defusing bombs, operating at hazardous locations like Fukushima, or performing work in remote locations like the deep sea. But even the industrial robots, which sometimes are blamed for replacing human labor for the purpose of cost efficiency, are quite beneficial at a close look:

- They can be applied for multiple type of tedious and repetitively tasks, which humans

unwillingly do.

- They lower production costs and therefore keep production site in high labor costs countries and save the remaining jobs.
- They are able to perform tasks with higher accuracy and hence deriving better quality products.
- They empower the production to be more complex since they operate faster and thus perform more processing steps in the same time.
- They keep the quality repetitively on the same level without fluctuations.
- They keep the materiel usage on the same level.
- They can perform harmful tasks like coating the car with paint.

There are multiple categories of industrial robots. The one type, important for this work, is the *"articulated robot"*.This robot type is made of 6 orthogonal mounted joints. In contrast to most other robot types, like for example the SCARA or cylindrical robots, which are only able to reach a point from a fixed direction, the 6 joints of the articulated robot allow the robot to approach a single point from different directions, as illustrated in figure ***(Fig. 1.1)***. The direction from which the robot is approaching a point is called "orientation" and is usually described by 3 angles (Rotation about X: γ,Rotation about Y: β,Rotation about Z: α) between the active reference frame and the wrist of the robot. The point itself is described by its coordinate (X,Y,Z). Therefore, each robot configuration can be described

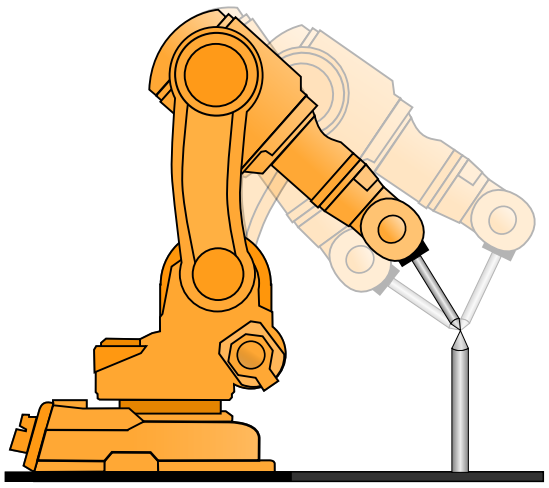

Figure 1.1: Different possibilities to reach a single point

by a 6-dimensional vector, which includes the orientation and the position. This vector is called a robot **pose** and is described by a transformation. There are multiple ways to describe such a transformation and, unfortunately, they can easily be confused. For industrial robots

however, the RPY angles are the most common. Details about transformations and their representations are in the appendix *[C.2]*.

In the most basic case, the robot's pose describes the relation form the robot's base frame[4], which is the coordinate frame originated close to the robot's base plate, to its wrist. An additional coordinate frame, called the tool-frame, is used to propagate from the wrist, which is usually the flange plate of the robot, to the mounted tool. If one considers the situation illustrated in figure ***(Fig. 1.1)***, he would receive the exact pose where the peak is pointing, if the robot's controller knew about the tool transformation of the mounted peak.

To determine the robot's wrist location related to the robot's origin, the robot controller calculates the so called *"forward transformation"*. The forward transformation can be calculated by considering each joint orientation, which is well-known due to the incremental encoders within the joints' engines and the physical length of the joints. This calculation can be executed by applying a special calculation scheme called the *"Denavit-Hartenberg-Transformation"*[3][5]. This scheme describes a standardized way to set up the transformation matrices based on a set of parameters *"Denavit-Hartenberg-Parameters"*), describing the certain characteristic of the robot's joints. Nowadays it is the standard way to calculate the forward kinematic for almost all industrial robots.

Having this forward transformation, the robot now knows where its wrist is positioned within its base coordinate frame. By knowing the tool transformation, it also knows where the tool is pointing relative to the base coordinate frame. Finally, it is quite common to provide a further information to the robot controller, which is the portion of the "part" the robot needs to apply its work on. This relation is often called the *"workobject"*[6]. If all these transformations are determined correctly, the robot control will be able to calculate the robot's tool-position within the coordinate space of the part. Thus, the robot knows where it operates relative to the part coordinate frame. Figure ***(Fig. 1.2)*** illustrates all these relationships. By applying such a complex system of transformations, the robot is always aware of the position of the tool center point (TCP) related to the workobject. Therefore, one will be able to plan the robot programs on a CAD model of the workobject, since the robot can use the transformations to guide its TCP exactly to the preplanned coordinates. Hence, one major benefit is being able to prepare program "offline" within a simulation and then load it into the robot. The alternative way is to block the robot for hours, sometimes even days, and create the programs "online" within the physical scene.

Another benefit is that the robot and the tool can be changed. If, for example, the tool breaks and the robot gets a new tool that is basically the same but has slight deviations, one only needs to determine the tool transformation and all the programs will perform as before. This is also true if the whole robot needs to be exchanged. After measuring the "workobject"

[4] Depending on the robot manufacturer this coordinate frame might be named differently (e.g. KUKA called it RobRoot)

[5] Sometimes abbreviated by DH-transformation

[6] Unfortunately there is no unique naming for most of the transformation between the robot manufacturers. Hence the "wokobject" is common in ABB controls, whereas KUKA calls it the "base frame". Similar situation like with the name of the origin above.

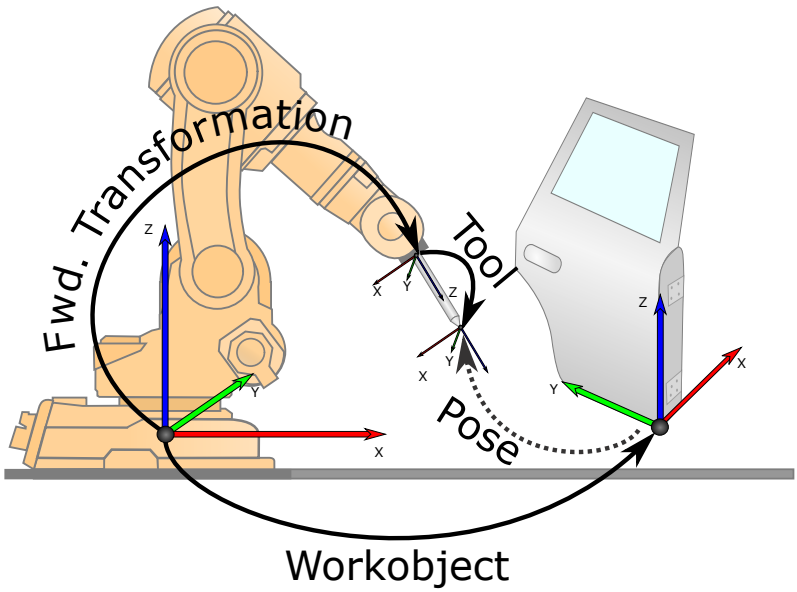

Figure 1.2: Relation between the most common frames for an industrial robot

transformations for the new robot, all the programs will run again the same way as on the old robot.

And, last but not least, having all these complex transformations allows the robot to determine the path in between the points within its programs. As a result, it is possible to move the robot exactly in circular tracks or a linear track without training the robot all the points between the start and endpoint. This would not be true if the robot uses only the joint configuration for determining the positions within its programs.

Nevertheless, all these transformations are fixed within the robot. Once the robot is configured and enters the automatic mode, the transformations cannot be changed. So, if the scene is not static and the transformations within the robot's workspace need to be adjusted, one needs to enhance the robot with systems that are able to determine the new configurations, measure the changes, and correct the transformation frames within the robot controller. Such systems are vision systems. They have a visual perception provided by sensors or cameras and provide the calculation to transform these measures into new frames for the robot.

1.1.2 Production in modern car plants

For complex goods like cars, the production is also separated further into several departments like:

- Press shop, where the plain metal sheets, arriving as coils, are cut and shaped;

- Body shop, where the raw body is assembled by welding and gluing the separate parts together;
- Paint shop, where the assembled body will be coated with multiple layers of functional and esthetic coatings; and
- Assembly shop, where the coated body shell will be further assembled with functional and interior parts until it is completed.

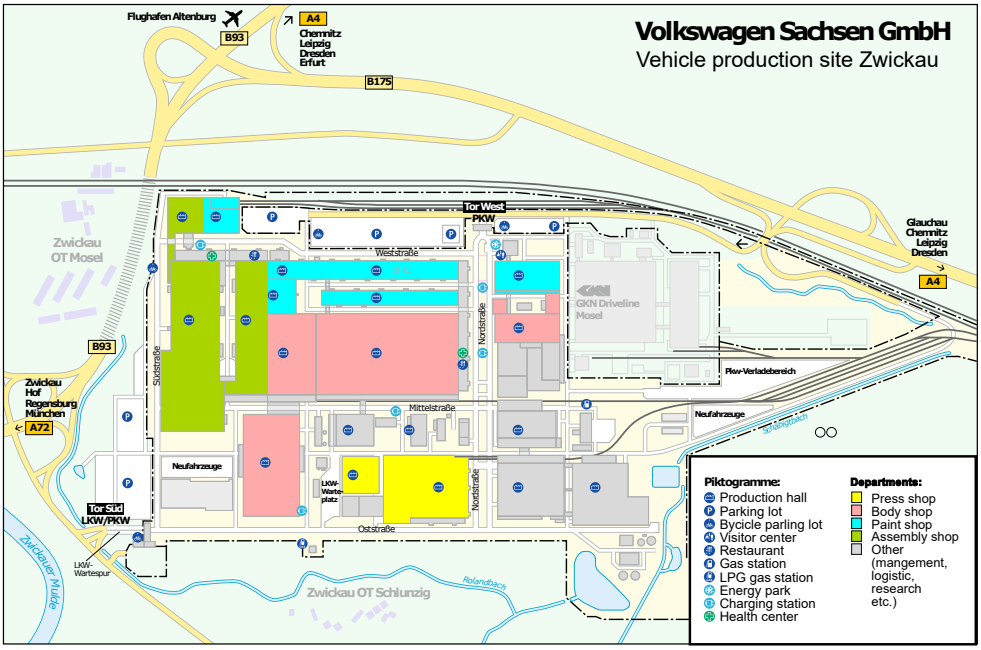

Figure 1.3: Shows a map of the Volkswagen plant in Zwickau Germany with colored production departments (reproduced with permission of Volkswagen AG)

Figure ***(Fig. 1.3)*** depicts the map of the Volkswagen production site in Zwickau Germany, where all the above-mentioned departments have been colored. They gray buildings are not related directly to production, but are for administrative, logistic or research purpose. Since the workstations are lined up sequentially along the assembly line[7], a workstation that is out of order can shut down a whole production line. Hence, a single workstation failure can cause a significant collapse of production output. And if one thinks of the number of workstations involved in a complex production process, it is easy to imagine that this can happen quite often. Therefore, each department has a well-trained maintenance team that takes care of the workstations. But still, every 3 to 5 minutes of line stop is equivalent to one less car in production output. Since such a variance in the production output would be unacceptable, there are production buffers in between the departments. Such buffers sometimes provide

[7] In rare cases workstation are build in parallel.

capacities within the quantity of a whole shift in order to compensate the missing output of a line stop. In the later stages of production, the buffers consist of large-capacity, automated storage in the form of a high rack that keeps assembled car bodies in multiple layers. Image***(Fig. 1.4)***[8] illustrates such a storage for painted car bodies within the paint shop.

And of course, the buffers are highly automated. The car bodies are not moved around by

Figure 1.4: A high rack storing multiple painted car bodies.

fork lift, but automatically put on line. Such automated processes can be found everywhere within modern production lines. And, as described within the section on industrial robots (section *[1.1.1]*), such machines, which include robots, have no perception of the environment and therefore are only able to handle static scenes if they were not equipped with some kind of sensor vision. Hence, they depend on static, non-changing, environments. And therefore, it is natural that one tries to set up automatic workspaces with fixed references between the automation units. In many situations this is perfectly possible. In the body shop, for example, the pressed steel sheets will be assembled to a whole car body. During this procedure, they will be fixed within fixtures in defined positions. Image ***(Fig. 1.5)***,for example, illustrates a fixture for a door frame in the body shop. Within such workspaces, the robot can work without problems since it only needs to go through the same fixed procedures over and over again. But while the car gets further assembled, it eventually will not be possible to put it into tables with fixtures to have a defined position. The car will be transported on a different kind of assembly belts and remains on those while the robots will perform their tasks. One such transportation system is called *"skid conveyor"*. There, the car will be transported by a construction called *"skid"*. The skid is kind of a slay-like metal frame on which the car sits. On the bottom side of the skid there are rail-like constructions. These rails run on conveyor belts with rolls that push the skid forward. Since the rails have excessive clearance on rolls, stopping positions of such conveyor belts are somehow inaccurate. A further problem is that the skids circulate within production, so they get old and twisted before they are changed. And last but not least, there usually is very poor contour accuracy of the skids even when they are new.

[8] Taken from a DURR commercial booklet of the EcoEMOS system at FAW/Volkswagen in Changcung China. (http://www.durr.com/fileadmin/user_upload/fas/EE_FAW-VW_Changchun_d.pdf)

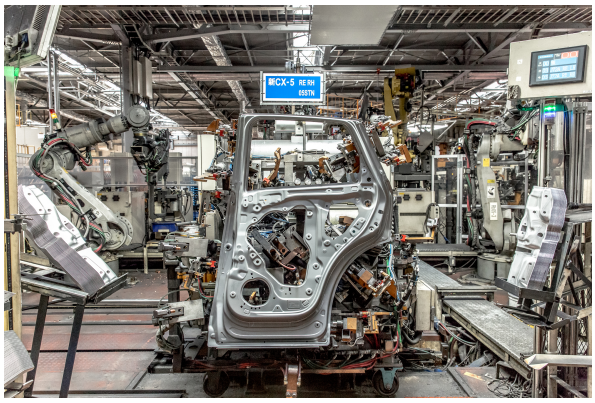

Figure 1.5: A door fixture within the body shop.

Another system is the *"overhead conveyor"*, where the car hangs in an overhead skid. Just as with the normal skid conveyor, these skids get worn out over time, the arms get bent, the rolls get worse, and stopping positions become inaccurate. Both conveyor types are illustrated

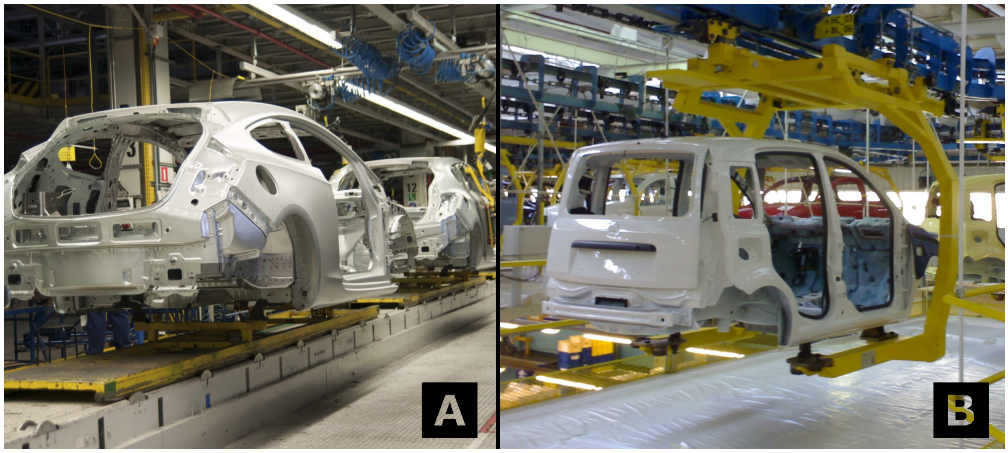

Figure 1.6: Image A: Car bodies on a skid conveyor. Image B: Painted body on an overhead conveyor

in figure ***(Fig. 1.6)***. The left image **(A)** shows an example of the *"skid conveyor"* and the right image **(B)** depicts an *"overhead conveyor"*. The two conveyor types from figure ***(Fig. 1.6)*** are the two most common types within automatic robot cells, but there are many other types, especially within the assembly shop.

However, the two conveyor types can easily have position inaccuracy of $\pm 15mm$. For some applications, such as painting, this might be acceptable. But for most of them this positioning

tolerance is far for the expected accuracy for most robot applications, which is usually under $1mm$. In these situations the robot need guidance to keep its tolerances within the requested limits.

1.2 Discussed topics and related chapters

The goal of this thesis is to develop new ideas and algorithms which permit path corrections on sensitive spots, like visible sealing beads on wing parts. For wing parts, such as doors, the challenge is not only that the door as a part itself is moveable, but it is also less rigid than bigger parts in the car body. This property also applies to multiple other wing parts like the fender, trunk lid, or the engine hood. Also, the missing rigidity of most of the wing parts makes it necessary to provide more than just a single correction vector.

Another challenge is that most of the sealing beads are positioned close to reference edges, which requires very high application accuracy. This is due to the fact that the human eye is capable to see even the slightest deviations as soon as there is a valid reference close. Image ***(Fig. 1.7)*** depicts this fact. Figure ***(Fig. 1.7)*** illustrates two line pairs. One pair of

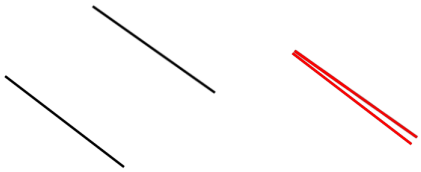

Figure 1.7: Visual effect of non parallel lines. Both line pairs are the same. The only difference is that the red lines are closer to each other.

black lines and another of red lines with exactly the same angle between the lines in both pairs. However, the average human would recognize the black lines as parallel, whereas no one would say that for the red lines.

Unlike most sealing beads, the path correction application spots are visible to the end customer and, even worse, most path correction tasks are related to a visual edge as well. As an example, one can consider a sealing bead at a crimped door edge. The edge at the door will be perfectly straight and thus similar as on the righthand side in ***(Fig. 1.7)***, so a human will be able to see the tiniest irregularity within the bead. Hence the need for accuracy for the correction vector is very high and, since most parts like doors are not very rigid, it is usually not sufficient to provide a single correction for a part, but rather multiple corrections along the contour lines. Therefore, the path correction technology proposed in this thesis provides multiple correction vectors for different regions on the part. The difficulty, however, turned out to be that it is hard to achieve a sufficient amount of correction vectors with respect to low numbers of features available on most of the wing parts. This is due to the fact that most wing parts are parts of the visible, outer body of the car, which tends to have only weak features like body lines. Strong visible features like holes or corners, which might serve as

a vision features, are unlikely to appear on wing parts like doors or trunk lids in sufficient amount. Therefore, most of the systems available today apply some kind of laser line sensors to be able to work with the body shape instead of visible features. Those sensors, however, have the drawback that they are only able to provide their results within the laser plane, which is two-dimensional. A similar problem appears within camera images, which also only provide a two-dimensional projection of the real world, and which will be discussed in detail in Section *[2.1.1]*. Unfortunately, the methods introduced in section*[2.1.1]* do not apply for laser line sensors. Hence, one of the key achievements of this work is the formulation of methods applicable for laser line sensors to perform three-dimensional measurement for position correction. This new methodology, based on [4], will be discussed in section *[3.3.1]*. Driven by detection faults within the laser line sensors on certain surfaces or spots, it is necessary to introduce a novel statistical method having the ability to detect and eliminate faulty image points. This method, which leads to more stable images, will be introduced in section *[3.1]*. And, finally, this thesis addresses the feature detection within the sensor images itself. Since there are multiple algorithms for point cloud matching but none of them fulfilled the performance requirements for the application within an inline path correction system, a new algorithm based on a special simplification of the sensor data will be presented in section *[3.2]*. Finally, in order to transfer the achievements made in path correction to the field of visual servoing, this work will also introduce a special one-step calibration for laser line sensors *[4.1]* and also a method to solve visual servoing tasks in a single step *[4.2]*.

Chapter 2

Technical foundations

This chapter is meant to give the reader an overview over the methods currently applied for camera and sensor based 3 dimensional measurements. It will concentrate on the two most common types of sensors, which are plain cameras and laser triangulation sensors. Both sensors come along with mathematical models to convert their images into a piece of information capable of guiding the robot during the application. Since laser triangulation is based on a camera, the principles of plain cameras, which will be covered in the section *[2.1]*, also apply for these types of sensors.

This section will cover the mathematical model behind a camera and how the two-dimensional image features will be transferred into the three-dimensional world. The understanding of these models is essential for the further understanding of this work. Not only to get an insight of how the laser line sensors are working, but also to understand the basic concepts of the path correction *[3]* and *[4.2]* which can be understood as an extension of the bundle adjustment and was originally derived from it, but also to be able to understand the principles of camera calibration which is a central part of the method introduces in section *[4.1]*. For the static calibration, as proposed in *[4.1]*, it is also necessary to understand the calibration of the lense distortion. Hence the modeling of a lens and its distortion, and how to determine the distortion model and eliminate it will be covered in section *[2.1.4]*. Section *[2.2]* will give some insights into how to calibrate all the internal and external camera parameters. Finally, section *[2.3]* will explain how to use all this information in order to perform measurements with the camera.

A second part of this chapter will cover the laser-based camera system. Section *[2.4]* will introduce the principle of laser triangulation and its underlying mathematics. Since there are different requirements for calibrating such sensors, section *[2.4.3]* will provide some insight about tool center point (TCP) calibration. This calibration need to be applied to get the tool frame of a triangulation sensor mounted on the robots wrist. The knowledge of the exact transformation is essential, to apply the the methods proposed in *[3]*. Without knowing the actual position of the measures the proposed path correction will not deliver correct results. Alternatively one could use the method as proposed in *[4.1]*, but in situations where it is not necessary to have a static calibration such an approach might be too complex.

2.1 Camera Model

Cameras are very common today and tend to appear in many situations. Starting from the observation and security cameras in public places to the small built-in cameras in devices of daily use like mobile phones or notebooks.

Within the automotive industries there are also a vast amount of applications around camera systems. Most of the time such tasks are pertain to inspection, such as observing a scenery and validating its correctness. But since cameras provide only a 2D image of the real world, one does not necessarily connect cameras to 3D measurements, which they are capable to do as well.

To enable 2D cameras to do 3D measurements, one needs to take the projection geometry and some prior knowledge into account. In other words, one needs to know what is going to be observed and how it will project into the camera image. Based upon this knowledge, the given image can be compared with the expected model of the projection. This chapter introduces the state-of-the-art methods to mathematically model a camera and its projection geometry. Hence, it will serve as a foundation to understand further techniques that are going to be introduced within this thesis later on.

2.1.1 Simple pinhole camera model

An ordinary camera just creates a two-dimensional image, which is a projection of the usually three-dimensional scene caught in this image. Therefore, one will not be able to extract clear three-dimensional information from the image, but it will at least be possible to model the projection geometry of the image-capturing device in order to calculate the ray trace of light captured at the image points. The simplest way to model such a projection geometry is the pinhole camera model. One can imagine the pinhole camera model as a light-tight box with an infinite small hole (aperture) on one side, as well as a light-sensitive film on the direct opposite site. Figure ***(Fig. 2.1)*** depicts such a pinhole camera. Since the rays of light are

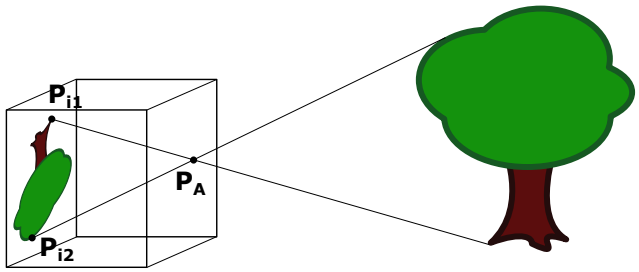

Figure 2.1: A simple pinhole camera

going to propagate linearly and the only possible point for the rays to enter the box is the aperture, they all have to intersect in this single theoretical point. Therefore, one can be sure that each ray visible on the film was going through this aperture point. And, therefore, one knows two well-defined points in room space, which are the image point P_{ix} on one hand

and the aperture point P_A on the other, which uniquely define a line representing the light ray's path through room space ***(Fig. 2.1)***. By having this information, one will still not be able to get the exact position of one point caught within the two-dimensional image, but at least one can be sure the imaged point is on the calculated ray, which reduces the possible positions in room space dramatically.

Finally, if some other well-known geometrical information about the observed object is also available, such as the object's height or coordinates of features within the object, one might be able to determine one unique position where the object fits the rays. Figure ***(Fig. 2.2)***

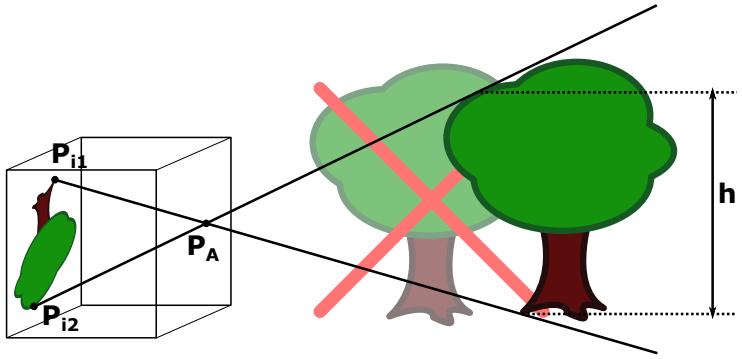

Figure 2.2: Illustrates that there is only one position, where the tree with given height fits on the rays.

depicts an example, where the height between two points is given as h. If one considers rotations impossible, there will be only one position where the object, in this case a tree, will fit in between the rays. Hence by taking some geometrical properties of the observed object into account, one is able to achieve detailed information of the object's position.

2.1.2 Thin lens model

In practice, the approach discussed in section *[2.1.1]* does not seem reasonable since common cameras are not pinhole cameras. Today's cameras come with usually complex lens systems in order to capture and focus a sufficient amount of light to achieve a good and quick exposure of the film or chip behind it. Hence, it is indispensable to purge from the idea of a small aperture which every ray goes through. The question arises whether the above-described model is still valid for today's cameras, since a larger aperture allows multiple rays to enter and the rays are not all going through the same point. To understand how the light propagates though a lens, one can apply the thin-lens model. This model serves a geometrical approximation of how light behaves when it encounters a lens. Figure ***(Fig. 2.3)*** shows a sketch of an ideal convex lens and how the light rays propagate through it. In ***(Fig. 2.3)***), the blue horizontal line separates the lens into two identically shaped parts. All rays, which are parallel to the optical axis before passing through the lens, will meet at the focal point **F** after passing through the lens. Figure ***(Fig. 2.4)*** illustrates this behavior. The distance between the focal point and the axis of the lens is called the focal length f ***(Fig. 2.4)***. The point behind the lens,

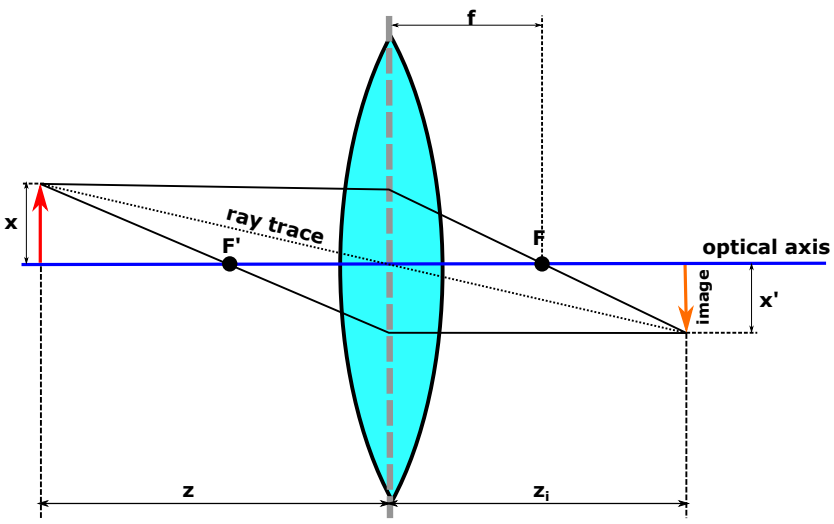

Figure 2.3: The scheme of an ideal lens

in which all rays coming from one object point converge, is called the **image point** ***(Fig. 2.3)***. The image point is the only point where the image of this object point is undistorted, since in every other position the rays do not converge and thus spread over a region of other neighboring image points. As figure ***(Fig. 2.3)*** depicts, one can calculate the position where the outgoing rays converge to the image point related to the focal length. Figure ***(Fig. 2.3)*** shows the two rays going to the focus point, one on the image side of the lens, and another through the mirrored focal point on the object side. Figure ***(Fig. 2.4)***) shows a scheme of the theoretical projection and its approximation by the ray traces. Hence, by knowing the focal length of a lens, one can easily calculate the image coordinates to expect by using the interception theorem.

$$\begin{aligned} & \frac{x}{f} = \frac{x'}{z_i - f} \\ \Leftrightarrow \quad & \frac{x}{x'} = \frac{f}{z_i - f} \end{aligned} \tag{2.1}$$

Therefore, the relationship between the focal length and the expected projection is given by equation ***(2.1)***. In equation ***(2.1)*** z_i denotes the distance between the camera chip, where the image of the real world is projected, and the lens. This distance is called the image distance. Whereas z, which stands for the distance of the real-world object to the camera, is called the object distance. The position x is representing the x-coordinate of a point on the real world's observed object, and x' marks the x-position of its related image point. Both distances are depict in figure ***(Fig. 2.3)***. Equation ***(2.1)*** shows how the ratio between the object's x-coordinate and the images x-coordinate is connected to the chip distance and the focal length. But usually one has no clue about the exact chip position and also the x of the object shall be unknown, since this is usually the quantity to determine by the camera measurement. Hence one needs to take a deeper look on the ray trace for further consideration.The ray trace is the direct connection between the objects point and the image point, as illustrated

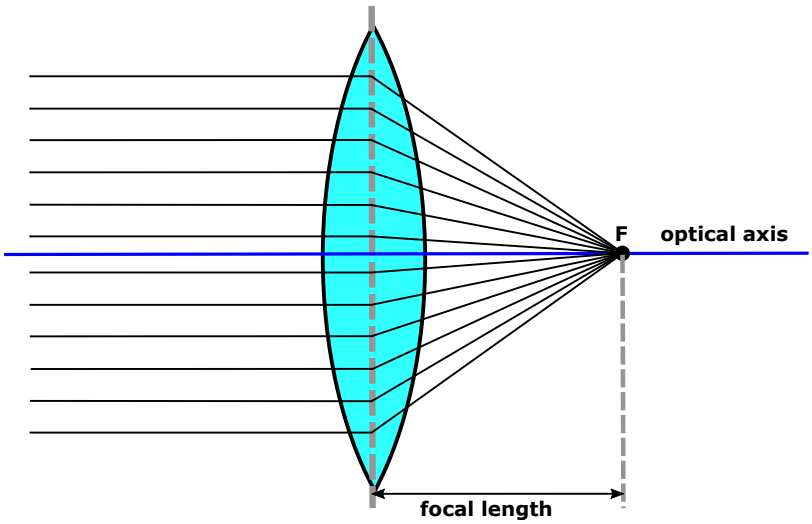

Figure 2.4: The focal point of an ideal lens

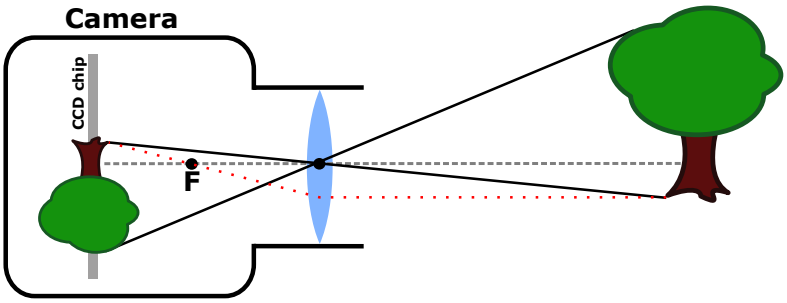

Figure 2.5: Raytrace and real projection

in figure ***(Fig. 2.3)***. By applying the interception theorem one more time, one can point out the following relationship:

$$\frac{x}{z} = \frac{x'}{z_i} \tag{2.2}$$

And by inserting the result of equation ***(2.1)*** into equation ***(2.2)***, one receives ***(2.3)***.

$$\begin{aligned}
\Rightarrow \quad & \frac{\frac{fx'}{z_i - f}}{z} = \frac{x'}{z_i} \\
\Leftrightarrow \quad & \frac{fx'}{z_i - f} = \frac{zx'}{z_i} \\
\Leftrightarrow \quad & \frac{z_i - f}{fx'} = \frac{z_i}{zx'} \\
\Leftrightarrow \quad & \frac{z_i}{fx'} = \frac{z_i}{zx'} + \frac{1}{x'} \\
\Leftrightarrow \quad & \frac{1}{fx'} = \frac{1}{zx'} + \frac{1}{z_i x'} \\
\Leftrightarrow \quad & \frac{1}{f} = \frac{1}{z} + \frac{1}{z_i}
\end{aligned} \tag{2.3}$$

Finally, the ray traces are going to intersect in a unique point, and thus can be used in the same way as the light rays within the pinhole cameras. By taking equation ***(2.3)*** into account, one can also determine the related geometrical properties between the image and the ray traces, and will be able to calculate them.

2.1.3 A camera model base on the thin-lens model

Although the mechanical construction of lenses can get really complex, especially in photography, the equations introduced in chapter *[2.1.2]* usually hold for a decent projection model.

And, if at all, they only need to be adapted for very special cases. Since in automation the expected scenes to observe usually are of a very constant nature, meaning that the observed object changes only at a small scale of millimeters, the lenses used are usually prime lenses. These prime lenses have a fixed focal length, which needs to be chosen in relation to the observed scene and, of course, in relation to the distance between the scene and the camera.

If one considers, for example, an object at a three-meter distance and a prime lens with the focal length of 50 millimeters, the sharp projection of the observed object will appear on the chip in the image distance z_i of 50.85 millimeters. This is due to equation ***(2.4)***, which is directly derived from ***(2.3)***.

$$\begin{aligned}
& \frac{1}{\frac{1}{f} - \frac{1}{z}} = z_i \\
\Leftrightarrow \quad & \frac{1}{\frac{1}{50} - \frac{1}{3000}} = z_i
\end{aligned} \tag{2.4}$$

Taking a closer look to the term $\frac{1}{z}$ of equation ***(2.4)*** clearly shows that the further the observed object gets from the camera, the closer the image distance gets to the focal length. Between ∞ to around 1 meter, the image distance usually varies between 1 and 3 millimeters[1]. Hence, common prime lenses can be adjusted a few millimeters, usually by turning, in order to have a sharp projection. On the other hand, one can easily see that the image distance is undergoing this change more rapidly, as soon as the obstacles to observe are getting very close to the cameras. This is especially an issue when it comes to so-called laser stripe

[1] Of course this is only approximately since it depends on the focal-length as well.

sensors, where a sharp projection over the full chip range is only possible by placing the chip at a certain angle, the so-called Scheimpflug angle.

On the other hand, if the object to observe is a decent distance away from the camera, one has a rough approximation, where he assumes the image distance to be approximative equal to the focal length.

$$z_i \approx f$$

If we go back to equation ***(2.2)*** and rearrange it to solve for x', and set the image distance z_i to the focal length, we will achieve the following equation for the x-position of the related image point:

$$f \cdot \frac{x}{z} = x' \tag{2.5}$$

Since equation ***(2.6)*** reproduces equally for y, one can generally state that any visible point $\underline{p} \hat{=} (x, y, z)^T$ projects like

$$(x, y, z)^T \rightarrow \left(f \cdot \frac{x}{z}, f \cdot \frac{y}{z} \right)^T \tag{2.6}$$

2.1.4 Lens distortion

Equation ***(2.6)*** would already hold for a good approximation for a camera model. But usually the light propagation through the lens is not as simple as the thin-lens model suggested. Usually, because of the physical shape of the lens, one can expect the image to be slightly distorted due to the lens's physical characteristics. This type of distortion is usually called spherical aberration. This distortion usually grows proportional to the focal length. The distortion is usually propagating radially from the optical center to the outer edges. Because of this, it is usually necessary to also determine the exact optical center, which depends upon the mechanical construction of the lens and the camera. Depending on the type of lens, the distortion can expand or shrink the image slightly, from the optical center outwards. If the

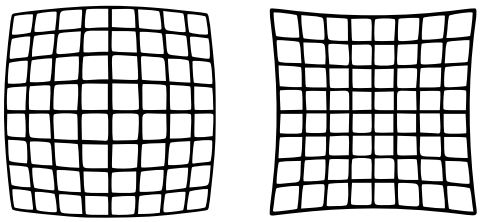

Figure 2.6: Sample radial distortions. Barrel distortion on the left and pincushion distortion on the right

distortion in the image tends to radial grow from the center, one talks about a pincushion distortion. If the image shrinks radially, it is called a barrel distortion. In special cases, there might also be some kind of tangential distortion, which usually tends to appear when there is

an angle between the camera chip and the lens. This might appear within devices, which aim to have a special focal plane by this constructive change. This kind of distortion is illustrated in image ***(Fig. 2.7)***. This kind of distortion appears especially within laser line sensors,

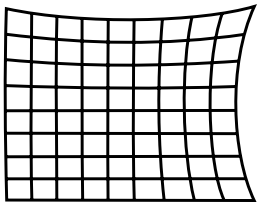

Figure 2.7: Tangential distortion

where the chip and the lens appear to be constructed with a certain angle - the so-called Scheimpflug angle [5], in between each other. The most common model addressing all these different kinds of distortions is the Brown-Conrady model [6] & [7]. The Brown-Conrady model is formulate in the two equations ***(2.7)*** and ***(2.8)***.

$$\begin{aligned} x_u &= x_d + (x_d - x_c)(K_1 r^2 + K_2 r^4 + \cdots) + (P_1(r^2 + 2(x_d - x_c)2) + \\ &\quad 2P_2(x_d - x_c)(y_d - y_c))(1 + P_3 r^2 + P_4 r^4 \cdots) \qquad (2.7) \\ y_u &= y_d + (y_d - y_c)(K_1 r^2 + K_2 r^4 + \cdots) + (2P_1(x_d - x_c)(y_d - y_c) + \\ &\quad P_2(r^2 + 2(y_d - y_c)2))(1 + P_3 r^2 + P_4 r^4 \cdots) \qquad (2.8) \end{aligned}$$

where

$$\begin{aligned} (x_u, y_u) &= Undistorted\,points \\ (x_d, y_d) &= Distorted\,points \\ (x_c, y_c) &= Center\,of\,distortion \\ K_n &= n^{th}\,radial\,distortion\,coefficient \\ P_n &= n^{th}\,tangential\,distortion\,coefficient \\ r &= \sqrt{(x_d - x_c)^2 + (y_d - y_c)^2} \end{aligned}$$

The parameters (x_d, y_d) are the distorted image points recorded by the camera and (x_u, y_u) are the resulting undistorted point coordinates. As indicated by the dots, this model can be set up to an arbitrary number of coefficients K_n and P_n. In practice though, it is very common to finish with K_3 and P_2. The model equation is highly non-linear, which makes it impossible to derive a close-form solution, or even an inversion, turning an undistorted point back into a distorted point. Therefore, in order to determine the distortion parameters vector $(K_1, K_2, K_3, P_1, P_2)$ and the related center point (x_c, y_c), one needs to apply iterative methods[2] converging to the final parameter vector.

The distortion calibration should be performed as a separate calibration step, before de-

[2] see chapter *[B.1]*

termining the extrinsic[3] camera parameter within a second calibration step. This is due to the fact that not all of the parameters of the two calibration steps are independent from each other. For example, changing the center point (x_c, y_c) of the chip is almost similar to moving the camera in the chip plane. Thus calibrating both parameters, chip center and extrinsic position, at the same time, can lead to an unstable Jacobian an therefore singularities. The distortion calibration should be performed as a separate calibration step, before determining the extrinsic camera parameter within a second calibration step. This is due to the fact that not all of the parameters of the two calibration steps are independent from each other. For example, the center point

In addition, the strength and type of radial distortion will influence the object distance parameter. Finally, it is important to note that the proportions of the parameters differ strongly. Hence the calibration appears to be a two-step procedure, where first the step is the calibration of the intrinsic parameters, distortion and its related center point [4], and the second is the calibration of the extrinsic parameters that are giving the camera position.

Having no extrinsic calibration also means the feature references are not useable during the distortion calibration. Therefore, one needs to fall back on geometrical shapes provided by the calibration body. Points that are linearly aligned[5] are very commonly used for that purpose, since the related equations are quite simple. The line points might be given by some features with line up in a pattern, like the code features in ***(Fig. 2.8)***) or the chessboard-like pattern applied within the OpenCv[6] calibration. The only important thing is to have some kind of features aligned in a matrix, where one can extract the edges, corners, or centers, and determine if they follow the virtual lines within the matrix[7]. The features can be detected without extrinsic calibration, they can be extracted within pixel space and they can be arranged to sets that represent the virtual lines. As a quality measure for a single line, one can apply the formulas given in equation ***(2.9)***.

$$\begin{aligned} \left[U, \Sigma, V^T\right] &= svd \begin{pmatrix} p_{1_x} & p_{1_y} \\ p_{2_x} & p_{2_y} \\ \vdots & \vdots \\ p_{n_x} & p_{n_y} \end{pmatrix} \\ \Delta d_n &= \frac{V \cdot \Sigma^{-1} \cdot U^T}{\left\| V \cdot \Sigma^{-1} \cdot U^T \right\|} \cdot \underline{p_n} - \left\| V \cdot \Sigma^{-1} \cdot U^T \right\| \end{aligned} \tag{2.9}$$

[3] Camera parameters are separated into two groups, which are the "intrinsic" and the "extrinsic" parameters. Intrinsic parameters are the internal camera parameters, like chip geometry and the projection geometry. Extrinsic parameters are the outer parameters, which describe the position of the camera in the scene. Detailed description can be found in section *[2.2]*

[4] There are more intrinsic parameters like focus or pixel size, details on that in section *[2.2]* .

[5] That means the points should be on a line.

[6] OpenCv is an free available image processing toolkit. It also provides modules for camera calibration. Details about the OCV calibration can be found under "https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html"

[7] Theoretically one can use other geometrical schemes, but matrix like schemes are by far the most common.

The operation "svd" in ***(2.9)*** stands for the "singular value decomposition" [8], which is used to calculate the pseudo inverse of the point matrix. The vector $\underline{p}$ is a vector of points extracted from the image, which should be arranged on a line, and $\overline{\Delta}d_n$ is the distance of the n^{th} point p_n to the best fit line through the whole set $\underline{p}$. Hence, ideally all elements of the vector $\underline{\Delta d}$ should be zero. The function presented in ***(2.9)*** can serve as an error function for a non-linear solver such as described within section *[B.1]*. But since the proportions of the distortion parameter $(K_1, K_2, K_3, P_1, P_2)$, which are usually in the range of $1e^7$ per pixel, and the center point, which is $1:1$, are so much ill-posed, there needs to be an adjusting factor applied to ensure the relation in between.

Image ***(Fig. 2.8)*** presents the resulting image after applying the above-described methods to a strongly distorted image containing a calibration plate on the left and the resulting undistorted image on the right. The green boxes are displaying the undistorted reference for the center region. Within the undistorted image on the right hand, one can clearly observe that the calibration marks at the edge of the green rectangle stay aligned to the reference edge, whereas on the left-hand side, those boxed in the center of the edges strongly exceed the limits.

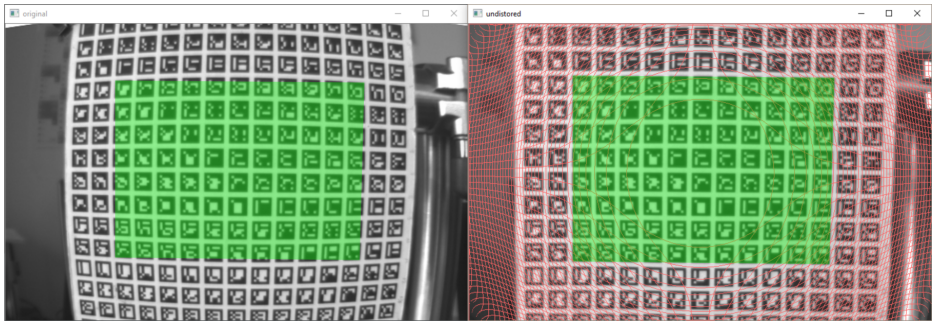

Figure 2.8: The strongly distorted image on the left and the resulting undistorted image after applying the Brown-Conarady model

2.2 Camera calibration

As described in the previous section, the calibration of the distortion is the first step and is usually applied by linear structures within the image. After accomplishing the distortion calibration, the nonlinearities have been removed from the system and the remaining projection can be described as a linear model. This linear model is set up by the equations describing the pinhole camera from section *[2.1.1]* and the lens projection introduced in *[2.1.2]*. Yet, these parameters do not cover the complete requirements of a camera calibration, since they only describe the internal projection behavior. Therefore these parameters are referred to as

[8] An SVD decomposition factorizes any matrix into the 3 unique matrices $\left[U, \Sigma, V^T\right]$, which contain both sides singular values and their scales. In this cased it is used to provide a numerical stable way to calculate the pseudo inverse of a matrix [8].

the *"intrinsic"* parameters of the camera. In order to be able to use the camera to perform measurements within the real world, it is important to know where the camera is located in the world coordinate space. Otherwise one would be able to calculate the ray trajectories of the light going through a pixel, but could not use it since one does not know where the ray originated and how it propagates though the world frame.

Therefore, the so called "extrinsic" parameters, which store the position of the camera, are also an important part of the camera calibration which needs to be determined. Such transformations are usually composed of two parts: an orthonormal rotation matrix R of size 3×3 and the translation vector t^T of size 1×3. A detailed description of transformation matrices can be found in the appendix *[C.2]*

Therefore a camera calibration must obtain those two sets, the "intrinsic" and 'extrinsic" parameters, as a result. A common way to obtain those parameters is the method of "direct linear transformation" (DLT), published by [9] and [10]. Unfortunately the method proposed in [9] only works with non-planar calibration targets, whereas the method presented in this work by Tsai [11] can also be applied to planar targets, which are more common. This method starts with a linear model, as given as in equation ***(2.10)***.

$$\begin{aligned} \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} &= \mathbf{K}\,[\mathbf{R}\ t] \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \\ \Rightarrow \lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} &= \mathbf{K}\,[\mathbf{R}\ t] \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \end{aligned} \tag{2.10}$$

where

$$\begin{aligned} u &= \frac{x'}{z'} \\ u &= \frac{y'}{z'} \\ \lambda &= z' \end{aligned} \tag{2.11}$$

In equation ***(2.10)*** u and v are the pixel coordinates, λ is the scale factor, which turns the projection coordinates into homogeneous coordinates and the vector $(x, y, z, 1)^T$ represents the homogeneous reference coordinates from the calibration target. Matrix $\mathbf{K}$ is holding the intrinsic parameters of the camera and is composed as given in equation ***(2.12)***

$$\mathbf{K} = \begin{pmatrix} f & 0 & x_c \\ 0 & f & y_c \\ 0 & 0 & 1 \end{pmatrix} \tag{2.12}$$

Here f is the focal length and x_c and y_c are the center point coordinates, or often also called the "principal point". The extrinsic parameters of the camera are contained in the 3×4

matrix $[\mathbf{R}\ t]$, which is composed of a 3×3 rotation matrix $\mathbf{R}$ and a 3×1 vector containing the translations. Both matrices $\mathbf{K}$ and $[\mathbf{R}\ t]$ can now be multiplied to a single 3×4 matrix $\mathbf{P}$ called the camera matrix.

$$\begin{aligned} \lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} &= \mathbf{K}\,[\mathbf{R}\ t] \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \\ \lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} &= \mathbf{P} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \end{aligned} \tag{2.13}$$

If one now considers multiple well-known reference features on the calibration target $\mathbf{X}_i = (x_i\ y_i\ z_i)$ it follows that for each feature the following equality must hold:

$$0 = \mathbf{P} \cdot \mathbf{X_i} - \lambda_i \begin{pmatrix} u_i \\ v_i \\ 1 \end{pmatrix} \tag{2.14}$$

And since the matrix multiplication of ***(2.14)*** is happening row-wise, one could also write them separately as stated within the equations to .

$$0 = \mathbf{X_i}^T \underline{p_1} - \lambda_i u_i \tag{2.15}$$
$$0 = \mathbf{X_i}^T \underline{p_2} - \lambda_i v_i \tag{2.16}$$
$$0 = \mathbf{X_i}^T \underline{p_3} - \lambda_i \tag{2.17}$$

In equation ***(2.15)*** to ***(2.17)*** the elements p_1, p_2, p_3 are representing the row vectors of $\mathbf{P}$. Within the next step equation ***(2.15)*** to ***(2.17)*** are transformed in a way, that all unknowns, which are the elements of $\mathbf{P}$ and the scale factors for the projection λ_i, a gathered in a single vector, whereas the remaining elements are coped within a matrix.

$$\underline{\mathbf{0}} = \underbrace{\begin{pmatrix} \mathbf{X_i}^T & \underline{\mathbf{0}} & \underline{\mathbf{0}} & -u_i \\ \underline{\mathbf{0}} & \mathbf{X_i}^T & \underline{\mathbf{0}} & -v_i \\ \underline{\mathbf{0}} & \underline{\mathbf{0}} & \mathbf{X_i}^T & -1 \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ \lambda_i \end{pmatrix}}_{\underline{h}} \tag{2.18}$$

Equation ***(2.18)*** achieved major step forward by rearranging the equation into the general linear shape $\mathbf{A}\underline{h} = \underline{\mathbf{0}}$. The next step would be to follow this scheme for a set of multiple references For the camera matrix $\mathbf{P}$ we have 12 elements, but only 11 degree of freedom due to the invariant scale within the projective geometry. With each reference, we gain 3 degrees of freedom but we also add one unknown scale factor. Thus there is a minimum of 6 references in order to be able to solve the problem. Equation ***(2.19)*** provides the full scheme

for multiple references.

$$\begin{pmatrix}0\\ \vdots\\ 0\end{pmatrix} = \underbrace{\begin{pmatrix}\mathbf{X_i}^T & \underline{\mathbf{0}} & \underline{\mathbf{0}} & -u_1 & 0 & 0\\ \underline{\mathbf{0}} & \mathbf{X_i}^T & \underline{\mathbf{0}} & -v_1 & 0 & 0\\ \underline{\mathbf{0}} & \underline{\mathbf{0}} & \mathbf{X_i}^T & -1 & 0 & 0\\ \mathbf{X_i}^T & \underline{\mathbf{0}} & \underline{\mathbf{0}} & 0 & -u_2 & 0\\ \underline{\mathbf{0}} & \mathbf{X_i}^T & \underline{\mathbf{0}} & 0 & -v_2 & 0\\ \underline{\mathbf{0}} & \underline{\mathbf{0}} & \mathbf{X_i}^T & 0 & -1 & 0\\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots\\ \mathbf{X_i}^T & \underline{\mathbf{0}} & \underline{\mathbf{0}} & 0 & \dots & -u_n\\ \underline{\mathbf{0}} & \mathbf{X_i}^T & \underline{\mathbf{0}} & 0 & \dots & -v_n\\ \underline{\mathbf{0}} & \underline{\mathbf{0}} & \mathbf{X_i}^T & 0 & \dots & -1\end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix}\underline{p_1}\\ \underline{p_2}\\ \underline{p_3}\\ \underline{\lambda_1}\\ \vdots\\ \underline{\lambda_n}\end{pmatrix}}_{\underline{h}} \tag{2.19}$$

Since the scale along the rays in a projective geometry is arbitrary, there will be a solution satisfying the constraint $\|\underline{h}\|^2 = 1$ for a certain set of λ. Hence the solution can be found by solving the homogeneous least squares problem of equation ***(2.20)***.

$$\min_{\|\underline{h}\|^2=1} \leftarrow \|\mathbf{A}\underline{h}\|^2 \tag{2.20}$$

Now one can expand equation ***(2.20)*** and replace the matrix $\mathbf{A}$ by its eigenvalue representation $\mathbf{A}\underline{x}_{\sigma_{min}} = \sigma_{min}\underline{x}$ ***(2.21)***.

$$\begin{aligned}\|\mathbf{A}\underline{h}\|^2 &= (\mathbf{A}\underline{h})^T Ah\\ &= \underline{h}^T\mathbf{A}^T\mathbf{A}\underline{h}\\ &= \underline{h}^T(\sigma_{min}\underline{x}^T_{\sigma_{min}}\sigma_{min}\underline{x}_{\sigma_{min}})\underline{h}\\ &= \sigma^2_{min}(\underline{h}^T(\underline{x}^T_{\sigma_{min}}\underline{x}_{\sigma_{min}})\underline{h})\end{aligned} \tag{2.21}$$

In ***(2.21)*** σ_{min} represents the smallest eigenvalue and $\underline{x}_{\sigma_{min}}$ is the corresponding eigenvector. Putting this back into equation ***(2.20)*** and settings $\underline{h} = \underline{x}_{\sigma_{min}}$ one achieves equation[9] ***(2.22)***.

$$\min_{\|\underline{h}\|^2=1} \leftarrow \|\mathbf{A}\underline{h}\|^2 = \sigma^2_{min}(\underline{x}^T_{\sigma_{min}}(\underline{x}^T_{\sigma_{min}}\underline{x}_{\sigma_{min}})\underline{x}_{\sigma_{min}}) = \sigma^2_{min} \tag{2.22}$$

Therefore a solution of ***(2.20)*** can be found through the eigenvalues, since the minimum must be where $\underline{h}$ aligned with the smallest eigenvector of $\mathbf{A}$. It is also common practice to solve for the minimum by using an SVD decomposition. The SVD is a matrix decomposition, which decomposes matrix $\mathbf{A}$ into 3 matrices $\left[\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}^T\right] = svd(\mathbf{A})$ where $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$. The interesting property of an SVD decomposition in this case is that $\mathbf{V}^T$ contains the eigenvectors of $\mathbf{A}^T\mathbf{A}$ and the diagonal of $\mathbf{\Sigma}$ are the corresponding eigenvalues ordered by size $\sigma_{1,1} > \sigma_{2,2} > \dots \sigma_{n,n}$. Hence the minimum of equation ***(2.20)*** is where the vector $\underline{h}$ is equal to the last row of $\mathbf{V}$.

[9] Keep in mind, that the length of a eigenvector is 1 and therefore the square $x^T x$ is 1, as well.

Now, having the minimizing vector $\underline{h}$ containing all the unknown elements of the camera matrix we also have a result for $\mathbf{P}$ and the final question is, how can one decompose this matrix into the two separate matrices containing the intrinsic and extrinsic parameters. Since matrix $\mathbf{K}$ was defined as an upper triangular matrix (equation ***(2.12)***) and that the rotation matrix $\mathbf{R}$ is a 3×3 is an orthogonal matrix, the QR decomposition seems to be an appropriate method.

> If $\mathbf{A}$ is an $n \times n$ matrix then there is an orthogonal matrix $\mathbf{Q}$ and a right triangular matrix $\mathbf{R}$ such that
>
> $$\mathbf{A} = \mathbf{RQ}$$

Considering equation ***(2.13)***, the translations that are also contained within the resulting matrix $\mathbf{A}$ must be within the last row of $\mathbf{P}$. Hence the matrix $\mathbf{A}$ must be the left-hand 3×3 sub matrix of $\mathbf{P}$. Finally, since element $a_{3,3}$ of $\mathbf{A}$ might not be 1, every element of $\mathbf{A}$ needs to be divided by $a_{3,3}$ in order to achieve the structure as given in ***(2.10)***.

2.3 Measuring with cameras

To understand how it is possible to measure with cameras, one has to recall the camera model introduced in section *[2.1.2]*. It described the model of the linear light propagation through the camera. Hence, it is possible to calculate the ray of light and how it propagates though room space. This is due to the fact that there are two well-known points on the ray. First, there is the principal point, where ideally all rays go through and where the camera coordinate system has its origin. Second, there is the point where the ray hits the CCD chip. This point is represented by a pixel position within the image and needs to be extracted with the help of a feature detection algorithm. The ray going through these two points is the original path of the light from the obstacle to the pixel position. Therefore, one can be sure the desired position is on this ray, but still there is an infinite number of possibilities since it can be on any positive distance away from the camera.

To be able to fix the position on the ray, one needs additional information. In the simplest case, this might be another camera observing the very same position. This constellation is called a stereo system, or stereo vision. Now there are two cameras and two rays. If those rays were ideal, they would intersect at some position if they are not parallel. This position must be the desired point in room space. In reality, however, there are various noise influences and hence the rays will be skewed but come very close in this certain point. The target position is then assumed to be the center of the line segment orthogonal to both of the rays in the point where they are the closest. Since this distance is usually very small, at least when the cameras are well calibrated, this point will still be referred to as intersection point.

This constellation of two cameras observing the same spot is depicted in figure ***(Fig. 2.9)***.

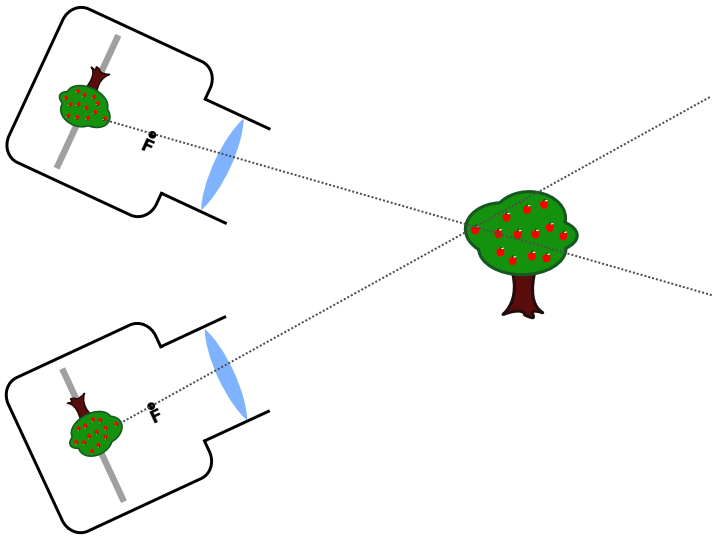

Figure 2.9: The two rays from each camera intersect in the real 3 dimensional position.

One can see the two cameras observing the apple and the schematic of the rays from the picture though the focal point and intersecting at the apple. Of course, such systems need both cameras to operate within the same coordinate frame. Therefore, they have to be calibrated with the same calibration target[10], or at least both target positions need to be related by an external measurement system in order to identify their position within the global measurement frame. The intersection point will then be a point within the three-dimensional calibration frame.

Another common measurement constellation with cameras is the so-called bundle adjustment. Bundle adjustment works with multiple features that are all located on a single object. Each feature will be extracted from the images with the help of an image processing algorithm and, again, the unique rays (bundles) will be calculated. For each feature one also needs to know the exact position within the object's frame. The object frame is the frame connected to the object. This is the frame which needs to be determined within the scene (e.g. world frame). Since the bundle adjustment determines not just a point, but a frame, it will also provide the orientation of the object.

Figure ***(Fig. 2.10)*** depicts such an arrangement. There are four cameras, each observing one feature. Two in the front and two in the back. The rays going through the features and the pixel within the image are indicated by the purple lines. The front features are marked with the red cross within the circle. One might be able to imagine that the car will not be able to move without pulling the features away from the rays. That means, on the other hand, that there is only a unique position for the car with the given rays. This position is the

[10] In this context same is also related to the position. The calibration plate should not move, while both cameras capture an image and perform the calibration

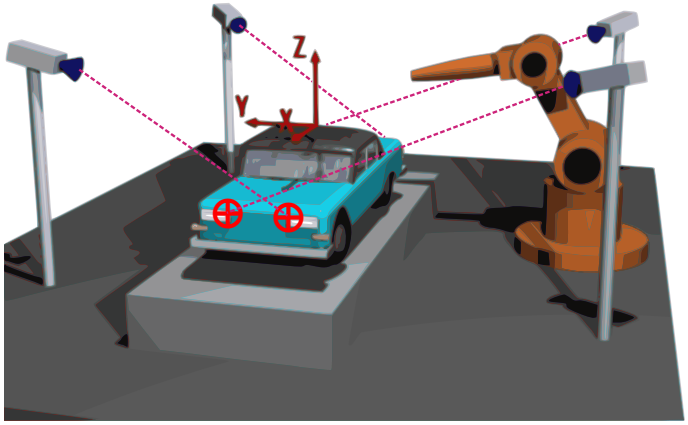

Figure 2.10: Bundle adjustment for car body detection.

result of the bundle adjustment.

If one considers n to be the number of known reference points in the object frame and m to be the number of cameras, then equation ***(2.23)*** yields the error function for the bundle adjustment.

$$\min_{\mathbf{X}} \sum_{i=1}^{n} \sum_{j=1}^{m} d(\lambda \mathbf{P}_j \mathbf{X} \underline{b}_i,\ \underline{p}_{i,j})^2 \tag{2.23}$$

Within equation ***(2.23)***, $\mathbf{P}_i$ is the camera matrix as derived in equation ***(2.13)*** in section *[2.2]*, mapping the known three-dimensional features $b_i = (x_i, y_i, z_i, 1)^T$ into the image plane. And finally $\mathbf{X}$ is the transformation which describes the cars position within the world frame. The function $d(\underline{p}_1, \underline{p}_2)$ determines the Euclidean distance between two pixel positions. Hence the final vector $p_{i,j}$ must be the extracted pixel position of the feature within the image. The equation ***(2.23)*** needs to be minimized with the help of some non-linear solver, like *"Levenberg Marquard"*. Further details about non-linear solving can be found in appendix *[B]*. Equation ***(2.23)*** just considers the linear projection. A more general approach is in equation ***(2.24)***, where $\mathbf{Q}()$ denotes a general function mapping the features into the image plane. This function might also include some non-linear image distortion as described within section *[2.1.4]*, as well.

$$\min_{\mathbf{X}} \sum_{i=1}^{n} \sum_{j=1}^{m} v_{i,j}\, d(\mathbf{Q}(\underline{a}_j,\ \underline{b}_i, \mathbf{X}),\ \underline{p}_{ij})^2 \tag{2.24}$$

Sine $\mathbf{Q}()$ will need the know the camera parameters for each camera, they will be supplied by the parameter vector $\mathbf{a}_j$. Finally, there is also a weight factor $v_{i,j}$. This can be used to reduce the influence of certain features, for example if the quality of the pixel position determined by the image processing is not very high, or just to turn off features which have not been detected.

The bundle adjustment can also be applied within camera calibration, with $m = 1$ providing an iterative scheme for just solving a subset of the camera parameters.

2.4 Laser triangulation for distance measurement

The path correction technology introduced within this work usually employs laser triangulation sensors to acquire the measurements. Such sensors are able to measure distances to object surfaces related to an internal sensor coordinate system. This is achieved using a laser projection observed by a built-in camera. The measurement principle of such sensors is called "laser triangulation" and there are two basic types of laser triangulation sensors.

Laser distance sensors measure only distances by projecting a laser dot on an obstacle. The reflection of the laser will be received by an array of small independent photo diodes, representing a row of pixels. Image ***(Fig. 2.11)*** depicts such an arrangement. In figure ***(Fig.***

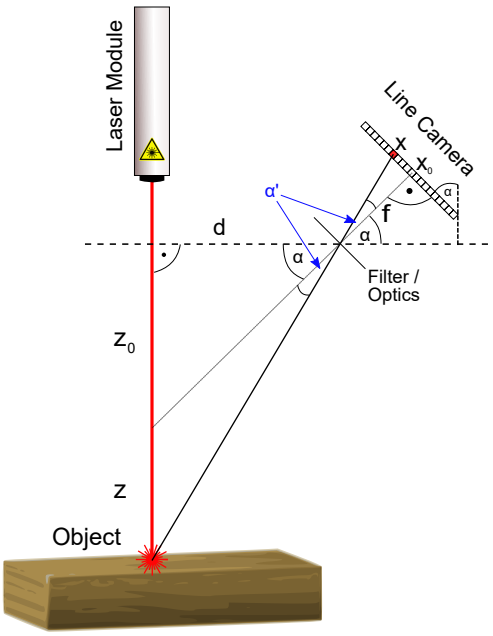

Figure 2.11: Laser triangulation principle

2.11), one can determine the desired distance z if the angle α, the distance d and the focal length f are fixed and well-known by a calibration method. Therefore, equation ***(2.26)*** which describes the major triangle, and equation ***(2.25)***, describing the minor triangle, need to be

combined into equation ***(2.27)***, which gives the distance z as a function over x.

$$tan(\alpha') = \frac{x}{f} \tag{2.25}$$

$$tan(\alpha + \alpha') = \frac{z_0 + z}{d} = \frac{tan(\alpha) + tan(\alpha')}{1 - tan(\alpha) \cdot tan(\alpha')} \tag{2.26}$$

$$z(x) = d \cdot \frac{f \tan(\alpha) + x}{f - x \cdot tan(\alpha)} - z_0 \tag{2.27}$$

2.4.1 Laser-stripe sensors

The basic construction of laser-stripe sensors is similar to that of the simple laser distance sensors described in the previous section. The major difference is that the whole concept from above is extended by one dimension. That means that the laser is not just projecting a point, but a line. And the sensor is not just an array of pixels, but a matrix like the chips in normal two-dimensional cameras. Figure ***(Fig. 2.12)*** depicts such an arrangement. The laser stripe

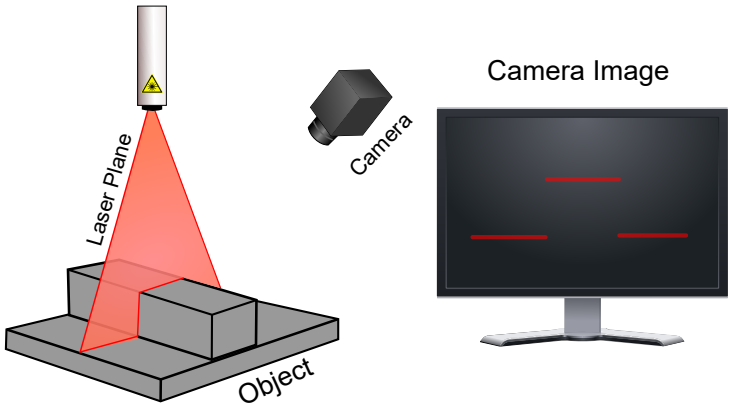

Figure 2.12: Illustrates the combination of a laser stripe generator, a camera and the resulting image

sensor projects a line on the object underneath. Where the laser meets the obstacle, a light reflection will be visible in the camera image. Hence, the resulting camera image must look like the image depicted within the monitor on the right side. Furthermore, it is clear that any possible reflection of an obstacle underneath the laser must be within the plane indicated on the left side outgoing from the laser. Therefore, the resulting image will always be a vertical section of what lies underneath the laser.

If the sensor is calibrated, meaning the extrinsic parameter between camera and laser as well as the intrinsic parameter of the camera are well-known [11], it is possible to transform pixel positions within the camera image back to real-world measures within the sensors coordinate system. This is possible because for every position within the camera image there is a

[11] Details about camera calibration and the parameters have been introduced in section *[2.2]*.

unique ray intersecting the laser plane. Figure ***(Fig. 2.13)*** illustrates this working principle. The rays can be calculated with the methods introduces in section *[2.1.1]* to *[2.1.4]*. The intersection point, then, is the real-world coordinate, achieved by calculating the plane line intersection point. Therefore, the transformation between the camera and the laser plane must be determined during calibration.

Since the laser line sensor employs a camera with a common lens, the lens distortion as described in section *[2.1.4]* applies 1:1 to the laser line sensor as well. To achieve high accurate measurements this distortion needs to be modeled and the parameter of the model need to be calibrated. A common way to calibrate such distortions based on a checkerboard, is described within the publication of Zheng and Kong [12]. Another publication employing a linearly moved target is [13].

But such a complex model is usually not necessary since, due to the static arrangement of

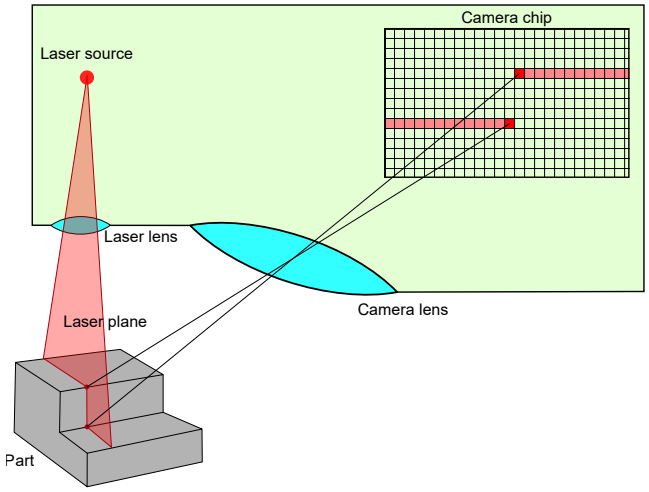

Figure 2.13: Illustrates the working principle of a laser stripe sensor

the laser and the camera, there is only one intersection point for each pixel. In one image the virtual ray of light from the pixel though the focal point can only intersect the laser plane in one place. Therefore, the simplest way to calibrate a laser stripe sensor is to position a target in well-known positions within the laser plane. The target may have a triangular shape, as depicted in figure ***(Fig. 2.14)***. For such a shape it is very easy to extract a point of reference at the peak, as shown with the two intersecting lines in the top image. But other shapes are also be possible, as for example spherical objects, where the center might hold for a stable reference point.

Finally, the real-world position of each pixel within the camera image can be calculated since the object's position is well-known. Hence, if the object is moved in such a way that the majority of pixels have been covered, one will be able to create a matrix which contains the 1:1 relationship between pixel coordinates and the real-world measure. If there are still holes left within this matrix, or if, to save memory usage, the matrix was chosen to cover only a

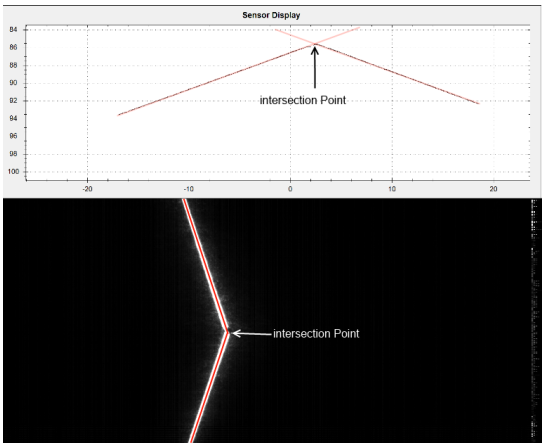

Figure 2.14: Top: Calibrated distance measures. Bottom: Raw camera image.

fraction of all pixels, the remaining pixel positions can be interpolated. The final result will be a look up table containing a depth value for each or a fraction of each pixel position.

The *look up table* described above also compensates for the lens distortion. But finally, it is not always possible to position targets very accurately within the laser plane and there are several publications[12] that offer different methods for solving the relation between the pixel potions and the laser plane, which are out of the scope of this work.

2.4.2 Sparsity of the measures

Although a laser-stripe sensor produces a cloud of three-dimensional measures, all measures are within a plane and therefore two-dimensional. The data is, however, different from camera images because the data is rather a vertical intersection than a projection. But, finally, there is still a complete degree of freedom missing within the images. If one considers for example the setup depicted in figure ***(Fig. 2.15)***, one might be able to imagine that the image of position 1 and position 2 will almost result into the same sensor image. There is obviously no information about the vertical position of the door without considering a position like position 3.

In multiple situations, especially when one wants to make measures within an image, the orientation of the measurement plane related to the contour is also an essential information. Figure ***(Fig. 2.16)*** depicts such a setup. In the left image, the laser plane is orthogonal to the surface and thus measures height relative to the surface. Within the right picture, there is a certain angle between the laser plane and the surface, which leads to a different height value. Hence knowing the position relative to scene is an important information. One has to recall that such devices are often built to measure within the 10^{th} of a millimeter and therefore the correct setup also plays an important role.

[12] Like [14] or [15]

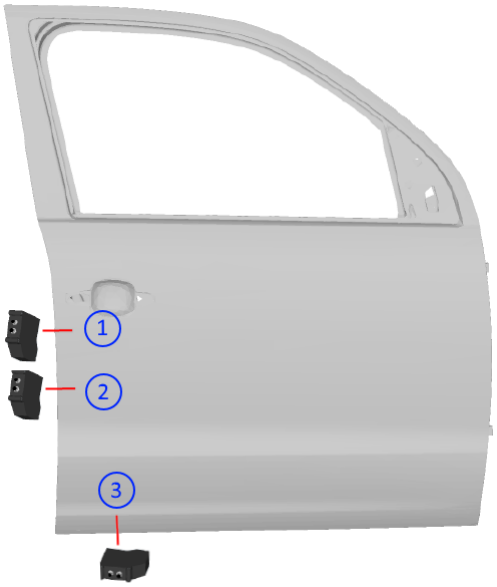

Figure 2.15: multiple measurement positions all having similar results

2.4.3 Sensor TCP calibration

The sensor coordinate system is determined within the calibration process of the sensor. Unfortunately, this coordinate system represents a type of virtual zero position related to the calibration process. For many sensors, it slightly varies due to production variations and variations within the calibration process. This might not be too consequential if the sensor is only used to measure distances within its own coordinate space, such as an object's height or gaps. On the other hand, if one imagines that multiple sensor images will be combined, or one or several sensors will be mounted on the robot wrist in order to provide information within the robot's coordinate space, then it becomes clear that it is important to know exactly the coordinate system of the sensor and its relation to the related components.

Finally, a major question discussed within this chapter is how to determine the position of the coordinate frame of the sensor, which is defined by the sensor's extrinsic parameters. Furthermore, since the major topic is path correction, we are interested in finding the relation between the robot's wrist and the sensor frame. If this distance is known, one can relate the measured points to the robot's work object, which is usually the ideal position of the object being measured.

One of the first publication about "Hand-Eye calibration" [16] by Zhuang, Roth and Sudhakar adresses a similar problem, but in the case they describe, the sensor was a camera. Since the camera is providing a fully qualified transformation if used with a calibration target (see *[2.2]*), this calibration can be done in a single step. But, in Aristos and Tzafestas analy-

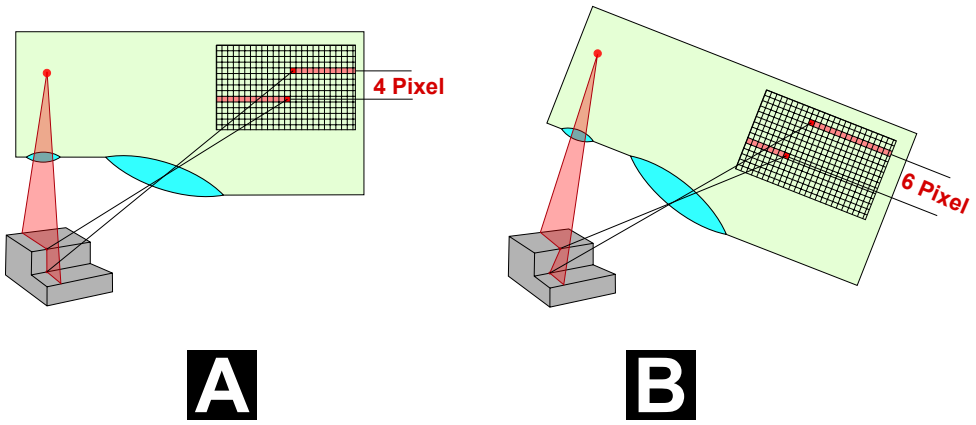

Figure 2.16: Measuring the box height from different angles

sis [17][13] of this method, they highlight that the robot causes the main inaccuracy in this process. There are similar methods described for laser stripe sensors, as described in [18]. Finally, these methods are strongly affected by the fact that the laser-stripe sensor is not able to achieve direct measures outside it's laser plane. All the information about the unobserved degree of freedom are based on geometrical assumptions, which affect the observable degrees of freedom. If one considers a slope, for example, the measured height can provide an insight about the position on this slope. But, as one can easily imagine, such indirect measures are not very accurate. Such methods usually work with structured bodies, where one can obtain the 3D position by analyzing the structure properties of the body from the given viewpoint.

A much better approach is to use a simpler body but multiple robot poses. This approach is schematically depicted in figure ***(Fig. 2.17)***. Here the calibration target is a sphere, which has a simple and distinct mathematical description. The robot is moving around the calibration target and records multiple images with the laser-stripe scanner. All poses from the robot base $\mathbf{T}_{tool_i}^{base}$ to the robots flange, where i is the i^{th} pose of n calibration poses, are well-known and can be supplied by the robot controller. Due to having multiple poses, the expected error will shrink since it will converge to an optimal average with the rising number of calibration poses.

Since one can expect each point p_i, within the sensor image, to be a measure of the spherical calibration target, one will be able to describe the target's equation as a given ***(2.28)***.

$$\mathbf{r}^2 \quad = \quad (c_{s_x} - p_{x_i})^2 + \left(c_{s_y} - p_{y_i}\right)^2 + (c_{s_z} - p_{z_i})^2 \tag{2.28}$$

Here r is the radius of the calibration target and $c_s = (c_{s_x}, c_{s_y}, c_{s_z})^T$ is the center point of the sphere. Since all points p_i have been measured with the sensor, the described center point c_s is also related to the sensor, which is denoted by the subscript s.

Therefore, equation ***(2.28)*** can be rewritten as a linear least square problem[14], which re-

[13] [17] is not only analyzing the accuracy, but there is also a closed form extension of the original method.
[14] Details about how to solve linear least square problems are laid out in the appendix *[A.2]*

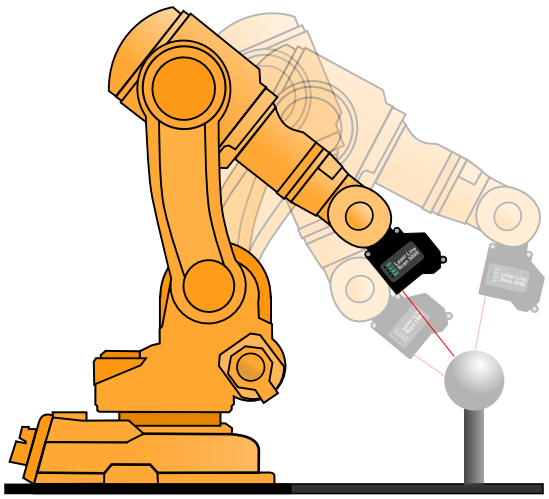

Figure 2.17: Procesure of laser stripe calibration

sults into the center point of the calibration sphere ***(2.29)***.

$$\mathbf{r}^2 = c_{s_x}^2 - 2c_{s_x}p_{x_i} + p_{x_i}^2 + c_{s_y}^2 - 2c_{s_y}p_{y_i} + p_{y_i}^2 + c_{s_z}^2 - 2c_{s_z}p_{z_i} + p_{z_i}^2$$

$$\Rightarrow \begin{pmatrix} p_{z_1}^2 + p_{y_1}^2 + p_{x_1}^2 \\ p_{z_2}^2 + p_{y_2}^2 + p_{x_2}^2 \\ \vdots \\ p_{z_n}^2 + p_{y_h}^2 + p_{x_n}^2 \end{pmatrix} = \begin{pmatrix} -2p_{x_1} & -2p_{y_1} & -2p_{z_1} & 1 \\ -2p_{x_2} & -2p_{y_2} & -2p_{z_2} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ -2p_{x_n} & -2p_{y_n} & -2p_{z_n} & 1 \end{pmatrix} \begin{pmatrix} c_x \\ c_y \\ c_y \\ c_x^2 + c_y^2 + c_z^2 - r^2 \end{pmatrix}$$

$$\begin{pmatrix} c_x \\ c_y \\ c_y \\ c_x^2 + c_y^2 + c_z^2 - r^2 \end{pmatrix} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{b} \tag{2.29}$$

with

$$\mathbf{A} = \begin{pmatrix} -2p_{x_1} & -2p_{y_1} & -2p_{z_1} & 1 \\ -2p_{x_2} & -2p_{y_2} & -2p_{z_2} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ -2p_{x_n} & -2p_{y_n} & -2p_{z_n} & 1 \end{pmatrix}$$

$$\mathbf{b} = \begin{pmatrix} p_{z_1}^2 + p_{y_1}^2 + p_{x_1}^2 \\ p_{z_2}^2 + p_{y_2}^2 + p_{x_2}^2 \\ \vdots \\ p_{z_n}^2 + p_{y_h}^2 + p_{x_n}^2 \end{pmatrix}$$

The basic idea is now that this center point of the sphere will be transformed into the "world" space of the robot. Since the calibration sphere is mechanically fixed, the center of the sphere

must always be in the same place.

In order to transform the sensor measures into the world space, one needs to take a closer look at the transformation chain illustrated in image ***(Fig. 2.18)***. It illustrates all relations between the frames and the sphere for a single robot pose. Following the scheme given in

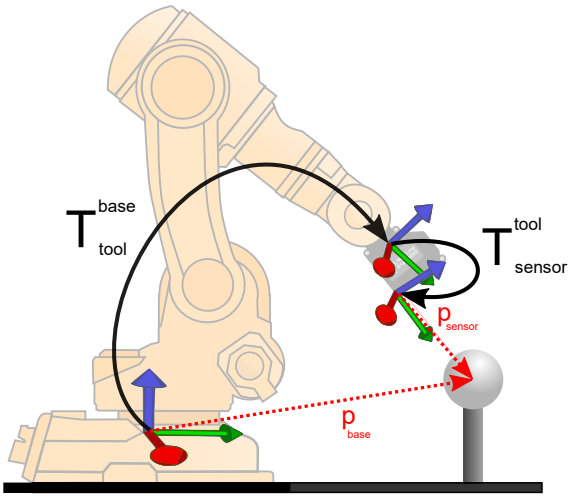

Figure 2.18: Relation of the transformation for a single calibration pose

(Fig. 2.18) we can set up formula ***(2.30)***. Within ***(Fig. 2.18)***, the transformation from robot base to tool $\mathbf{T}_{tool}^{base}$ is well-known and can be extracted from the robot controller for each point.

$$p_{base} = \mathbf{T}_{tool}^{base} \cdot \mathbf{T}_{sensor}^{tool} \cdot \mathbf{p}_{sensor} \quad (2.30)$$

$$p_{base} = f_{base}^{sensor}\left(\mathbf{T}_{sensor}^{tool} \middle| \mathbf{T}_{tool}^{base}, \mathbf{p}_{sensor}\right) \quad (2.31)$$

$\mathbf{p}_{sensor}$ are the points measured by the sensor and thus are also known, which leaves us with $\mathbf{T}_{sensor}^{tool}$ as the only unknown quantity within this equation. Hence ***(Fig. 2.18)*** will be rewritten as a function of $\mathbf{p}_{sensor}$ ***(2.31)***.

Finally, since there is a function to transform the measured points into the world frame, which in this case is the robot base, we can transform all points measured by the sensor. Once all points are transformed, they can be put into formula ***(2.29)***,since those points must be a sphere. Therefore, going from equation ***(2.29)***, one needs to insert for $\mathbf{A}$ ***(2.32)*** and

for $\mathbf{b}$ ***(2.33)***.

$$\mathbf{A} = \begin{pmatrix} -2f_{base}^{sen.}\left(\mathbf{T}_{sens}^{tool}\left|\mathbf{T}_{tool_1}^{base},\mathbf{p}_{sens_{1_1}}\right.\right)^T & 1 \\ \vdots & \vdots \\ -2f_{base}^{sen.}\left(\mathbf{T}_{sens}^{tool}\left|\mathbf{T}_{tool_1}^{base},\mathbf{p}_{sens_{n_1}}\right.\right)^T & 1 \\ -2f_{base}^{sen.}\left(\mathbf{T}_{sens}^{tool}\left|\mathbf{T}_{tool_2}^{base},\mathbf{p}_{sens_{1_2}}\right.\right)^T & 1 \\ \vdots & \vdots \\ -2f_{base}^{sen.}\left(\mathbf{T}_{sens}^{tool}\left|\mathbf{T}_{tool_m}^{base},\mathbf{p}_{sens_{n_m}}\right.\right)^T & 1 \end{pmatrix} \tag{2.32}$$

$$\mathbf{b} = \begin{pmatrix} f_{base}^{sen.}\left(\mathbf{T}_{sens}^{tool}\left|\mathbf{T}_{tool_1}^{base},\mathbf{p}_{sens_{1_1}}\right.\right)^T f_{base}^{sens}\left(\mathbf{T}_{sens}^{tool}\left|\mathbf{T}_{tool_1}^{base},\mathbf{p}_{sens_{1_1}}\right.\right) \\ \vdots \\ f_{base}^{sen.}\left(\mathbf{T}_{sens}^{tool}\left|\mathbf{T}_{tool_1}^{base},\mathbf{p}_{sens_{n_1}}\right.\right)^T f_{base}^{sens}\left(\mathbf{T}_{sens}^{tool}\left|\mathbf{T}_{tool_1}^{base},\mathbf{p}_{sens_{n_1}}\right.\right) \\ f_{base}^{sen.}\left(\mathbf{T}_{sens}^{tool}\left|\mathbf{T}_{tool_2}^{base},\mathbf{p}_{sens_{1_2}}\right.\right)^T f_{base}^{sens}\left(\mathbf{T}_{sens}^{tool}\left|\mathbf{T}_{tool_2}^{base},\mathbf{p}_{sens_{1_2}}\right.\right) \\ \vdots \\ f_{base}^{sen.}\left(\mathbf{T}_{sens}^{tool}\left|\mathbf{T}_{tool_m}^{base},\mathbf{p}_{sens_{n_m}}\right.\right)^T f_{base}^{sens}\left(\mathbf{T}_{sens}^{tool}\left|\mathbf{T}_{tool_m}^{base},\mathbf{p}_{sens_{n_m}}\right.\right) \end{pmatrix} \tag{2.33}$$

When inserting ***(2.32)*** and ***(2.33)***, the result is the theoretical sphere center for a given sensor mounting position $\mathbf{T}_{sens}^{tool}$.Since this mounting position is the unknown quantity, it will be the parameter vector of the non-linear optimization function as given described in appendix *[B]*. Finally, the last step is to derive the objective function, which is given in equation ***(2.34)***.

$$\epsilon\left(\mathbf{T}_{sens}^{tool}\right) = \begin{pmatrix} \left\| f_{base}^{sen.}\left(\mathbf{T}_{sens}^{tool}\left|\mathbf{T}_{tool_1}^{base},\mathbf{p}_{sens_{1_1}}\right.\right)^T - (c_x,c_y,c_z)\right\|_2^2 - r \\ \vdots \\ \left\| f_{base}^{sen.}\left(\mathbf{T}_{sens}^{tool}\left|\mathbf{T}_{tool_1}^{base},\mathbf{p}_{sens_{n_1}}\right.\right)^T - (c_x,c_y,c_z)\right\|_2^2 - r \\ \left\| f_{base}^{sen.}\left(\mathbf{T}_{sens}^{tool}\left|\mathbf{T}_{tool_2}^{base},\mathbf{p}_{sens_{1_2}}\right.\right)^T - (c_x,c_y,c_z)\right\|_2^2 - r \\ \vdots \\ \left\| f_{base}^{sen.}\left(\mathbf{T}_{sens}^{tool}\left|\mathbf{T}_{tool_m}^{base},\mathbf{p}_{sens_{n_m}}\right.\right)^T - (c_x,c_y,c_z)\right\|_2^2 - r \end{pmatrix} \tag{2.34}$$

2.5 Path correction

The path correction is separated in two phases, where the first is the measurement phase and the second is the correction phase. For the measurement, the robot positions the laser line sensor described in section *[2.4]* to the pre-defined measurement spots. To be able to do so, the sensor is mounted on the robot's wrist ***(Fig. 2.19)*** close to the application tool. As soon as the robot positions the sensor at a measurement spot, it will also trigger the sensor to record an image. Depending on the cycle time, the robot can stop at each measurement spot to make an image or trigger dynamically while moving over the spot. If the image is triggered during motion, it is necessary to ensure that the robot motion is linear[15] within the

[15]A linear robot motion"linMove" keeping same distance and orientation relative to the edge contour.

Figure 2.19: Robot wrist with sensor at roof-ditch application

region of the measurement spot. This is necessary since it is almost impossible for dynamic measures to trigger the sensor exactly in the right position. There will be all kinds of delays starting from cycle times of all CPUs employed within the communication up to the signal runtimes and repeating delays at bus interfaces. Hence, this will result into a trigger jitter, which, depending on the underlying bus system, can be quite big.

Thus, it follows that the measure at each spot provides no information in the direction of motion but still the remaining two degrees of freedom providing some information which will be used for further calculations. Therefore, it is very important to set the robot's path in such a way that the sensor images do not change within a certain distance around the measurement spot. If, however, the sensor images change due to the trigger jitter, the algorithm will calculate slightly different correction values for the very same position. Therefore, setting up the system carefully is crucial for stable results.

At least three measurements are necessary to calculate the correction values. These measures need to cover different orientations of the sensor. If there are numerous spots, the system can also provide multiple correction vectors, which might make sense if the path is not of static shape but provides slight deformations. Figure ***(Fig. 2.20)*** illustrates such a situation.

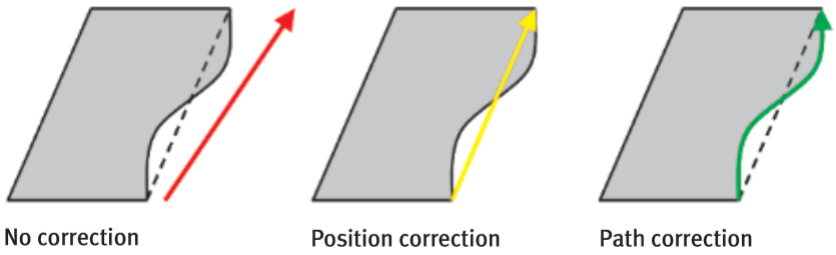

Figure 2.20: Difference between position correction and path correction

In the first sub-image, it shows the red arrow, which was the original path. It followed the outer right edge of a part. For the actual part, displayed in gray, this edge exhibits some deformations since its outer shape is curved and the part is not in its original position as well. If there is only one correction, the resulting path will end up as illustrated by the yellow

arrow. It will be at the right spot, but must follow the different outer shape. The green arrow illustrates what will happen if there are multiple corrections available. Deformation within the outer shape will be balanced since the correction vector can change during the application.

2.5.1 Panel fitting Systems

Vision systems for panel fitting are designed as an open control loop, guiding the robot iteratively until it achieves a final good position. In general, it would be desirable to have the panels fit in a single step, but due to missing features on most of the car bodies' panels there is no globally applicable fitting algorithm available yet. Hence, iterative control loop-based systems are primarily in used for panel-fitting applications. One example for such processes is the positioning of moveable panels into a car body. One can consider a front door as an example for such a panel.

A front door in a common production line will exhibit slight deformations and the same thing happens with the door frame where it needs to be installed. Furthermore, both the door and the car will not be in perfect position on the production line. After all, the part needs to be placed relative to the body. That means the vision system needs to determine both the frame where the part needs to be placed and the part itself. Thus, such systems have to deal with multiple unknowns, since a position needs to be determined which is not related to any absolute point in room space, but relative to the door coordinate frame predefined within the robot's cell.

On the other hand, the demand for accuracy is very high since there is only a small gap between the door and its frame and a customer can notice even slightest asymmetries [16]. Therefore, such systems usually are built with multiple sensors located around the parts to provide live measures during the robot's movement. The robot's position will then be controlled by the system until all sensors measure the desired values. The heuristics applied to control the robot are usually based on sensitivity matrices. Such matrices can be generated within the process or primarily set up in a training process[19].]. A common type of sensor for such systems is usually a combination of a laser and a camera acting as a laser-stripe sensor, similar to the sensors introduced in sectio *[2.4.1]*.

Hence, one will achieve a two-dimensional scan of the underlying objects shape. Sometimes these sensors are combined as double heads. This means that two sensors are combined in a way that the angle in between the sensors is about 90 degree and their laser lines overlap. This arrangement ensures that both sensors have a different view of the very same spot, so one achieves the same position from different view ports. This is especially beneficial in cases where the gap between the parts is high and deep, or the parts have overlapping structures.

For a door within a frame, the quantities targeted by the control loop are the gap and the flush values for each measurement spot. The gap is, as the name might already indicate, the gap between the frame and the door. The flush indicates how much the door overlaps the frame or dives into it. But both measurements are purely relative measures, providing

[16] This is because the frame serves as reference edge, similar as described in *[1.2]*

no information about where exactly these measures have been taken. To get a relationship between the sensor values and the robot's motion, a training process will be initiated. This training usually means that the robot is moving the part over a defined distance within the frame and records the behavior of the control values. Finally, one is able to say:

" by moving the robot about x mm in y-direction the flush in sensor 1 will change about k mm and the gap in sensor 2 will change about l mm".

These values, providing the change of the measures divided by motion of the robot, will be stored in a matrix. And, finally, the inversion of this matrix will give an insight about the necessary motion to perform to acquire a certain change within the sensors. Therefore, the recorded matrix is called a sensitivity matrix since it stores the sensitivities of the values related to the robot motions, and its inversion is called a Jacobian matrix [19]. Image ***(Fig. 2.21)*** depicts such a visual servoing system mounting a trunk lid. One can see that the sensors installed on the robot's grabber are surrounding the part and are arranged in a manner that both the trunk lid and the trunk lid's frame are in an observable control position[17]. As soon the robot reaches the control position, the control process is triggered and the sensor signal will be sent to the control PC, where new controls are calculated and sent back to the robot.

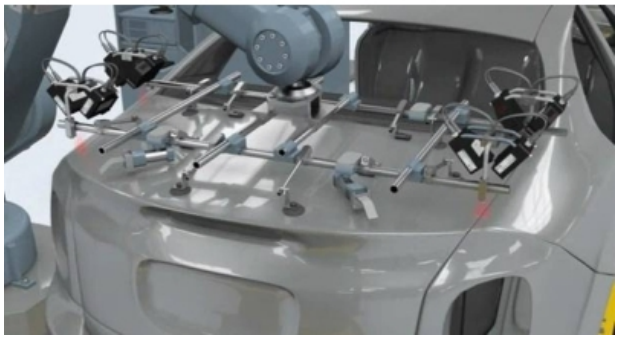

Figure 2.21: An illustration of the VMT-BestFit system mounting a trunk lid.

[17] The control position is a position close to the mounting position, providing some extra space for the robot control.

2.6 State of the art

As the title of this thesis implies, the main target of this work was to develop a path correction system for industrial robots. Hence this "state of the art" section is trying to give a general overview about the actual development and research of which most publication concentrate on visual servoing systems.
However, a more detailed overview, regarding the content of each research topic within this work, can be found in the introduction of the related chapters *[3.2]*, *[3.1]*, *[3]*, *[4.1]* and *[4]*.

The scientific research in the field of sensor based robot guidance and more specific in "visual servoing" dates back to 1979 [20], but is still very active today. "Visual servoing" systems utilize sensor generated data to actively correct the robot motion and are therefore frequently applied in path correction applications [21], [22] and [19]. These systems can appear in multiple configurations. Sensors can have a static position within the workspace [23],[24] and [25], or can appear in a so called "eye-in-hand" configuration where the sensor is directly mounted on the robots wrist [26] or [27].

However for any type of visual servoing system the aim is to minimize the error $\underline{e}(t)$ between the given $\underline{m}(t)$ measures and the expect values in a final target position $\underline{s}^*$. Which has been summarized in a general formulation ***(2.35)*** by Chaumette and Hutchinson [28].

$$\underline{e}(t) = \underline{s}(\underline{m}(t), \underline{a}) - \underline{s}^* \tag{2.35}$$

In ***(2.35)*** $\underline{m}(t)$ is a vector of features within the sensor image at a given time t. The function $\underline{s}(\underline{m}(t), \underline{a})$ converts the raw sensor information to "visual features" [28] which usually requires some sensor or environment specific parameters, represented by the vector $\underline{a}$.

In a very simple case the visual servoing can work with direct image features, which means the the features can directly be extracted from the sensor image. Such systems are called **IBVS** [18]. In this case ***(2.35)*** reduces to ***(2.36)***.

$$\underline{e}(t) = \underline{s}(t) - \underline{s}^* \tag{2.36}$$

Starting with the error definition of ***(2.36)*** one needs to find a scheme to improve the actual position $\underline{p}(t)$ of the robot, in order to reduce the error $\underline{e}(t)$ over time. The traditional concept for such a scheme is given in equation ***(2.37)***.

$$\dot{\underline{s}}(t) = \boldsymbol{L} \cdot \dot{\underline{p}}(t) \tag{2.37}$$

In ***(2.37)*** the matrix $\boldsymbol{L}$ is called the *feature Jacobian* matrix, which maps the speed vector $\dot{\underline{p}}(t)$ into the image space $\dot{\underline{s}}(t)$.

However, due to their simplicity and the limited capability of pure **IBVS** based approaches, a pure **IBVS** based correction is usually not appropriate to control a robot in all 6 degree of freedom, since for such complex cases a pure **IBVS** approach is prone to singularities and

[18] Image-based Visual Servoing

falling for local minima [29].

Nevertheless the are some really promising approaches where multi-view port systems, like stereo cameras, were successfully implemented for 6 dimensional correction. One of these publications is [30], where the authors were able to overcome the stability problems of **IBVS** in 6 dimensional space by introducing a "virtual visual space". This "virtual visual space" has been created based on the homography between the two view ports of the cameras and results into as virtual 3D space. Having such a virtual space, encoding the physical geometry of the two view ports solves a lot of stability issues.

Another interesting **IBVS** approach is proposed in [31], where the authors use a ANN[19] based approach. In [31] the authors created a four layer feed-forward ANN. Within their approach they track features of the target object while the robot is in motion. Similar to [30], they use a homography projection to map features into an "image feature space". This "image feature space" will then serve as an input for the ANN. The task of the ANN is to solve the complex an highly non linear mapping of the "image feature space" movements to robotic arm movements. By using an ANN the authors proposed a very efficient mechanism to learn the non linear relations between the joint in the real world and the related feature responses within the "image feature space".

An alternative approach for visual servoing systems are the so called **PBVS**[20] systems. Such systems determine the objects *real* pose in a 3 dimensional reference system by using the objects geometrical information, which is called the "object localization problem" [28].

Such **PBVS** based systems provide enhanced stability and high convergence speed, which is due to the fact that objects are located within a "real world" frame and not within some projection space. On the other hand they have the disadvantage, that geometrical information about the object to localize are necessary to solve the "object localization problem".

That means if the actuator, which is performing the task, knows his postion relative to the measurement frame with high accuracy and can also perform its motions with high accuracy, it is possible to approch the target position in a single step. By having such a configuration a control process based on sensor data is simply superfluous. Such a system is introduced in [32], where the authors are proposing a system for depalletizing and bin picking. For the object detection and localization the authors use surfels[21] coupled with a "soft assign' registration[22] approach. After determining the object the robot will be commanded to the target. In addition, to determine a collision free path, the authors also proposed a motion planing module.

A more general approach is proposed in Pfitzner et al [33], where the authors present a "object localization" approach combining multiple sensor data and types. As mentioned be-

[19] ANN = Artificial neuronal network
[20] Position-Based visual servoing
[21] surfel: **sur**face **el**ements, which are tiny planar elements detected within 3 dimensional point clouds.
[22] Soft assign is an extension of ICP where points have a weighted relation to multiple correspondences instead of a 1:1 relation. See section *[3.2]*

fore, the exact knowledge of the sensor position is crucial. Also in [33] the authors wrote, that one of the fundamental problems in the object localization were the inaccuracies caused by the moving sensors:

> Commonly, high precision is only given while moving the sensor with low speed or in deadlock. In addition, the acquisition of encoder and 3D data needs to be synchronized.

For the path correction, as it is presented within this work, the previous statement is no longer true. The proposed algorithm is designed to work with an "unobserved" degree of freedom, which is configured to be always in motion direction. Therefore the proposed path correction, as it is presented in section *[3.3]*, is also applicable in dynamic cases.

Chapter 3

A new approach for path correction

The major aim of this thesis is to derive a new type of path correcting system. This system should utilize laser-stripe sensors, which have been introduced in section *[2.4]*.Beside the improvement of the plain calculation of the correction vectors, it is also important to improve the data supplied by the sensors, as well as the methods of feature extraction.

Hence, as the first step, we will take a closer look at a statistical method for filtering and improving the data provided by the sensors. Section *[3.1]* will propose a novel technique to filter noisy sensor data. Despite the high capabilities of laser stripe sensors, these sensors become unstable when measuring in dark or strongly-reflective environments. Ambiguous points within a camera image can appear on dark surfaces and be confused with noise when the laser-reflection intensity approaches the noise level. Similar problems arise when strong reflections within the sensor image have intensities comparable to that of the laser.

Another major requirement is the detection of features within the sensor image. Flexibility is the major requirement for such an algorithm. Hence section *[3.2]* proposes a new algorithm to perform pattern matching on laser stripe sensor data. It is ICP-based, but unlike the ordinary ICP, it executes considerably faster. This is achieved by a smart reduction of features within the sensor data.

Finally, the information extracted about the feature locations within the sensor data, needs to be post processed to a correction vector. Therefore, section *[3.3]* introduces an algorithm capable of calculating corrections specialized on sparse measures. The sparsity is due to limited dimensionality of the laser stripe sensors, which had already been discussed within section *[2.4.2]*. Finally the algorithm proposed in section *[3.3]* outlines a solution for overcoming this problem and combine multiple measures to a complete correction vector.

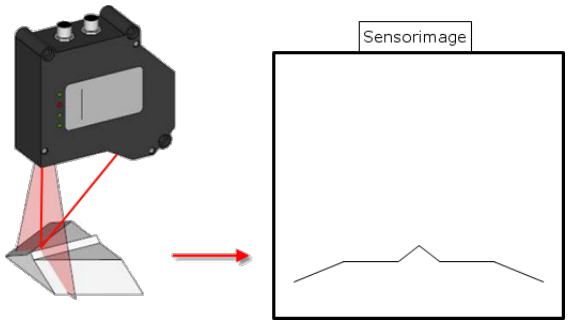

Figure 3.1: Laser-stripe Sensor. The bright red region indicates the laser curtain and the dark red line is the empirical path of a single beam.

3.1 A novel statistical method for noise filtering within the sensor data

Laser-stripe sensors are capable of measuring a vertical section of a surface related to an internal-sensor coordinate system using a laser-line projection observed by an internal camera. There has been a lot of research done on laser-stripe sensors, with one of the first publications introducing the measurement technique in the mid-eighties [34]. Currently, applications for such sensors are ubiquitous, covering engineering fields such as industrial automation, geodetic measurements, computer vision (3D modelling), and robot guidance. When laser-stripe sensors are properly calibrated, i.e. the extrinsic parameter between camera and laser and the intrinsic parameter of the camera are known, it is possible to transform pixel information of the laser-line reflection in the camera into real-world measurements within the sensor-coordinate space [35]. This is possible given that for every position within the camera image, there exists a unique ray intersecting the laser plane. This intersection point represents the real-world coordinate, which is computable if the camera calibration is known. To transform a coordinate from the image space to the sensor-coordinate space, a linear operation is applied and represented by a homography matrix [36, 37]. However, since the images are slightly distorted due to the lens system, it is necessary to model lens distortion in order to first untwist the images.
Various calibration methods for laser-stripe sensors have been published [38, 39].
Figure ***(Fig. 3.1)*** depicts a laser-stripe sensor observing a calibration body on the left-hand side and the associated measurement image on the right-hand side. Since the laser projection is observed by a camera, one fundamental task of such sensors is to extract pixel coordinates from the laser-line reflection within the camera image. Unfortunately, limited research exists concerning line-extraction from laser line sensor images. Currently, most commercial devices continue to use standard techniques introduced in the early nineties [40]. Small changes have been proposed, e.g. alternative methods for extracting the centre line of the laser [41]. Among the few works concerning image noise, W. Quing-Yang, et al., introduced a method to remove environmental noise by subtracting two images [42]. We still do not have methods for enhancing the stability of laser-line detection on dark or reflective surfaces involving dynamic

noise caused by the laser. Although strong laser reflections leave bright, easily extractable signals within the camera image, laser-line detection becomes challenging when additional reflections are visible within the image or, in the case of dark surfaces, if the reflection intensity approaches noise level.

Given the growing industrial popularity of laser-stripe sensors, applications involving complicated situations are becoming more common. Even in environments with strong reflective properties, the algorithm presented here can support line recognition in case of ambiguous situations involving contour gaps, jumps, or structural shadows. In such circumstances, an efficient method is required to determine the most probable candidate or decide whether a candidate exists. One common technique defines a threshold limiting the range for expected valid successors. If there are multiple candidates within the threshold, one strategy might be to consistently choose the neighbour candidate located closest to the latest laser reflection point. Unfortunately there is no equidistant spacing between samples and the closest point is not always the correct choice. In fact, the spacing between measured samples can vary due to the angular relation between the sensor and the object surface ***(Fig. 3.2)***.

One can expect the highest density of sample spots to exist where the sensor directly faces the object surface and lower sample density in regions where the angle between the surface and the sensor offsets from 90 degree. In order to address these situations, it is necessary to consider drift properties within the model equations. Here, we propose an algorithm based on geometric Brownian motion (GBM), an extension of the Brownian motion [43] used in modelling asset-price behaviour in mathematical finance [43]. However, since line-detection logic is an integral part of the real-time software controlling the sensor hardware[1], execution speed offers an additional challenge. This algorithm is intended to run on embedded systems in real-time environments, making the avoidance of complex calculations or iterative root-finding algorithms a major requirement.

An additional application for the proposed algorithm involves scan segmentation. Given that points of discontinuities are detected automatically, the algorithm can be applied to determine separate segments within a scene. This can be useful for filtering, since points of discontinuities usually constitute critical spots, or for object detection or separation, since one segment usually represents one object within an observed scene.

3.1.1 Working principle

The algorithm is arranged into two logical steps, where the first step determines whether or not a measurement is within the estimated range for a successor. This step is not limited to just one successor, but can determine parameters for an arbitrary number of successors. Therefore, step one verifies whether only a few successors exceed the estimated expected range by being classic outliers or if all upcoming successors are outside the expected range. In the event that there are many candidates for the next successor, the algorithm chooses the point with the highest probability, which is often the measure closest to the predicted mean. Points outside the limits defined by the predicted variance will be rejected as outliers.

[1] Usually an embedded system with only limited resources

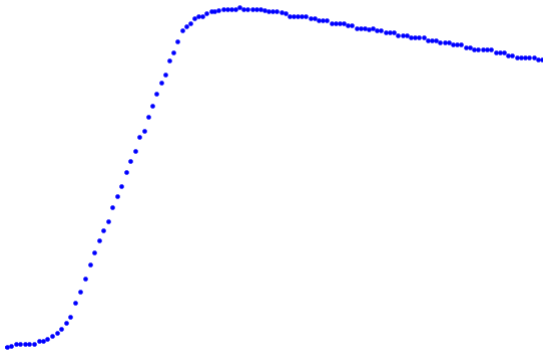

Figure 3.2: A typical car-body scan showing differences in sample densities.

The second step constitutes the update-step, wherein a maximum a posteriori (MAP)-estimator is utilized to predict the most probable measurement state, $\bar{s}_i$. The prediction of $\bar{s}_i$ is based on its actual measurement, s_i, and its previous state, s_0. Therefore, $\bar{s}_i$ serves as a recursive input for subsequent prediction steps.

To determine a segmentation point, the algorithm determines if a well-defined number of measurements violates the expected sample distance. If so, it is probable that these points are not outliers, but rather define a new segment of contour points. In this case, the algorithm needs to be reinitialized with the first contour point of the new segment. In cases involving outliers, these points will be removed from the dataset. For model determination, it is necessary to assess the nature of the distances between samples. Given the simplest case, where the distances are constant and with normally-distributed noise superimposed, one can establish an initial model as ***(3.1)***.

$$p_s(s_i|\ \bar{s}_i, \sigma^2) = \left(\sigma \cdot \sqrt{2\pi}\right)^{-1} \exp\left(-\frac{(s_i - \bar{s}_i)^2}{2\sigma^2}\right) \tag{3.1}$$

where ***(3.1)*** s_i denotes spacing *(sample distance)*[2] at the i^{th} measurement sample. The parameter, $\bar{s}_i$, denotes the expected spacing, predicted based on previous measures. Finally, σ^2 represents the variance between previous measures and their predictions. Thus, this initial model ***(3.1)*** represents a normal distribution, wherein both parameters represent the measurement expectation, $\bar{s}_i$, and the variance, σ^2, and are unknown and considered random variables. Therefore, the primary goal is to derive a model for these two parameters.
Starting with the spacing expectation, $\bar{s}_i$, it is possible to model it as its own random variable. Then, given a *PDF* for $\bar{s}_i$ relative to the given data, s_i, it is possible to solve for $\bar{s}_i$, such

[2]We use s *(spacing)* for the spacing between two samples instead of d, since d might be confused with the differential operator $\frac{d}{dx}$ later on.

that $p_{\bar{s}}(\bar{s}_i|\ s_i) \rightarrow max$. Therefore, in order to solve for the most probable $\bar{s}_i$, the first step is to derive $p_{\hat{s}_i}(\bar{s}_i|\ s_i)$.

3.1.2 A MAP estimator for $\bar{s}_i$

To find a proper model for the distribution of $\bar{s}_i$, it is necessary to define the spacing properties between measurements. The following sample-spacing properties for the *PDF* were defined as follows.

- The distances are always positive.
- Direction changes are random, but with drift taken into account.
- The drift influences the quantity of the spacing.

The direction changes described imply that distances between samples can grow or diminish, with either resulting in a specific direction. Without any additional information, there is no assumption of a preferred direction. The angle between the sensor and surface will influence sample spacing, which will also correlate to angular change and explains the necessity of the model to consider drift. Furthermore, it is expected that distances will rise more rapidly as the angle between surface and sensor increases. However, if the angle does not change, both directions will have equal probability.
A situation having equal probabilities for both directions is referred to as the *Wiener Process*. Due to the assumption that previous values can indicate a direction, the model also needs to take this drift into consideration. This leads to a process referred to as *Wiener Process with drift* or *GBM*. GBM is a method widely used in finance mathematics to model stock-price behavior [43, 44]. The *GBM* model equation for the i^{th} value of the spacing, s_i, and a given drift, μ, is expressed in equation ***(3.2)***.

$$ds = s_i\left(\mu di + dW_i\right) \tag{3.2}$$

Since the *GBM* is based on a *Wiener Process*, W_i, it is also a stochastic process. The *PDF* of the *GBM* results in a log-normal distribution Øksendal [45]. Equation ***(3.3)*** shows the *PDF* of the *GBM*.

$$\begin{aligned} p(s_i) &= \frac{1}{s_i\sigma\sqrt{2\pi i}}\exp\left(-\frac{\beta^2}{2\sigma^2 i}\right) \\ with \quad \beta &= \ln\left(\frac{s_i}{s_0}\right) - \left(\mu - \frac{1}{2\sigma^2}\right)i \end{aligned} \tag{3.3}$$

In equation ***(3.3)***, s_0 is the current measurement, where s_i is the i^{th} value ahead of s_0. By providing probabilities for future values, the *PDF* of equation ***(3.3)*** offers insight into the likelihood of a measurement belonging to the actual segment.
Thus, a first step in finding a measurement for the expected spacing between two samples is to use the *PDF* ***(3.3)*** to calculate the expected value, $\bar{s}_i$. The expectation of a sample, distributed according to equation ***(3.3)*** is given by equation ***(3.4)***.

$$\mathbb{E}(s_i) = s_0 \cdot \exp\left(\mu i\right) \tag{3.4}$$

The initial estimate of the distance can now be compared to the real measurement. In order to determine the certainty of this expectation, a confidence interval around the expected value needs to be calculated prior to the measures being validated. This interval will increase relative to the prediction supplied by equation ***(3.4)***. This uncertainty will be a linear combination of the expected variance for the i^{th} value of equation ***(3.4)*** and can be derived from the log-normal distribution in equation ***(3.3)***, resulting in the expected variance ***(3.5)***.

$$\begin{aligned} \mathbb{V}(s_i) &= s_0^2 \left(\exp\left(\sigma^2 i\right) - 1\right) \cdot \exp\left(2\mu i\right) \\ \hat{\sigma}(i) &= s_{i-1}^2 \left(\exp\left(\bar{\sigma}^2 (i-1)\right) - 1\right) \\ &\quad \cdot \exp\left(2\mu(i-1)\right) \end{aligned} \tag{3.5}$$

In equation ***(3.5)***, $\bar{\sigma}^2$ represents the recursive measurement of mean variation between the measurements relative to their expectation. This should not be confused with $\hat{\sigma}$, which is the uncertainty of the expected future mean calculated by the *GBM*. The variance, $\hat{\sigma}$, will be used to calculate the upper and lower envelope of the 95% confidence interval range, which is $\pm 2\hat{\sigma}_{i+k}$ around the expected value, $\bar{s}_{i+k}$ ***(3.4)***. A measurement within this 95% range will be accepted as valid and used to update the model parameter. However, if the actual measurement is outside the interval $\bar{s}_{i+k} \pm 2\hat{\sigma}_{i+k}$, it is rejected as an outlier. Given that this value might represent a single outlier, a defined number of successors also need to be tested. Since equations ***(3.4)*** and ***(3.5)*** are capable of predicting an arbitrary number of possible future values, the $(i+1)^{th}...(i+n)^{th}$ values can be calculated and tested. Large numbers of values not fulfilling the expectations are treated as a new segment, however, in the event that only a single value or a small number of values fails to meet expectations, these measurements will be treated as outliers.
If the actual measurement has been accepted, the density function, ***(3.5)***, and the actual measurement represent the prior distribution, allowing a MAP-estimator to update the estimation parameter for the following step.

3.1.3 Updating the model parameter

In the previous section, methods were derived to estimate successive values. To predict a range for acceptable measurements, the variance of the prediction was used. Given this range, it is possible to decide whether or not a measurement can be used for further calculations. The ability to eliminate invalid samples improves the model parameter by utilizing all remaining samples to estimate the *"optimal"* spacing, $\bar{s}_i$. This value should be the one value, $\bar{s} \in \mathbb{R}^+$, that maximizes the density function, $p(\bar{s}|s_i)$. To derive the density function, $p(\bar{s}|s_i)$, it is necessary to apply Bayes rule. This results in a MAP-estimator [46, 47], which estimates the most probable state for $\bar{s_i}$ based on previously collected data.

$$\begin{aligned} p(\bar{s}_i|s_i) &= \frac{p_N(s_i|\bar{s}_i, \sigma^2) \cdot p_G(\bar{s}_i)}{p_s(s_i)} \\ \Leftrightarrow \quad p(\bar{s}_i|s_i) &= \frac{p_N(s_i|\bar{s}_i, \sigma^2) \cdot p_G(\bar{s}_i)}{\displaystyle\int_{\bar{s}_i \in S} p_N(s_i|\bar{s}_i) \cdot p_G(\bar{s}_i)\, d\bar{s}_i} \end{aligned} \tag{3.6}$$

The numerator of equation ***(3.6)*** includes two density functions, where the first, $p_N(s_i|\hat{s}_i, \sigma^2)$, is given in ***(3.1)*** and the second, $p_G(\bar{s}_i)$, is given in equation ***(3.3)***. The denominator contains the normalizing integral, which limits the area of the resulting density function to one. Following the integration of $\bar{s}_i$, the expression becomes constant relative to $\bar{s}_i$ and thus can be removed from ***(3.6)***, given that a constant factor does not alter the extrema position. This results in equation ***(3.7)***, which provides a maximum threshold for the expected sample spacing from the i^{th} step.

$$\begin{aligned} \hat{s}_i &= \max_{\forall s_i \in \mathbb{R}} \left(p(\bar{s}_i|s_i)\right) \\ &= \max_{\forall s_i \in \mathbb{R}} \left(\frac{p_N(s_i|\bar{s}_i) \cdot p_G(\bar{s}_i)}{p_s(s_i)}\right) \\ &= \max_{\forall s_i \in \mathbb{R}} \left(p_N(s_i|\bar{s}_i) \cdot p_G(\bar{s}_i)\right) \end{aligned} \tag{3.7}$$

Combining a normal distribution with a log-normal prior results in a non-conjugate pair, meaning that the maximum threshold derived from equation ***(3.7)*** has no analytical solution. Therefore, the goal is to provide a reliable method for accurate approximation of the real maximum threshold for $\bar{s}_i$. Starting with the equation ***(3.7)***, it follows:

$$\begin{aligned} & \max_{\forall \bar{s}_i \in \mathbb{R}} \left(p_N(s_i|\bar{s}_i) \cdot p_G(\bar{s}_i)\right) \\ = & \max_{\forall s_i \in \mathbb{R}} \left(e^{\left(-\frac{(s_i-\bar{s})^2}{2\sigma^2}\right)} \cdot \frac{1}{\bar{s}_i} \cdot e^{\left(-\frac{\beta^2}{2\sigma^2 i}\right)}\right) \\ = & \max_{\forall s_i \in \mathbb{R}} \left(e^{\left(-\frac{(s_i-\bar{s})^2}{2\sigma^2}\right)} \cdot e^{\left(-\frac{\beta^2}{2\sigma^2 i} - ln(\bar{s}_i)\right)}\right) \\ = & \max_{\forall s_i \in \mathbb{R}} \left(e^{\left(-\frac{(s_i-\bar{s})^2}{2\sigma^2}\right)} \cdot e^{\left(-\frac{\beta^2 - ln(\bar{s}_i)(2\sigma^2 i)}{2\sigma^2 i}\right)}\right) \end{aligned} \tag{3.8}$$

The next step is to evaluate the numerator in the exponent of the second term in ***(3.8)***, from which some constants can be eliminated, given that they do not affect the maximum of $\bar{s}_i$.

$$\begin{aligned} & \beta^2 - ln\left(\bar{s}_i\right)\left(2\sigma^2 i\right) \\ = & ln\left(\frac{\bar{s}_i}{\bar{s}_0}\right)^2 - 2\alpha \cdot ln\left(\frac{\bar{s}_i}{\bar{s}_0}\right) + \alpha^2 + ln(\bar{s}_i)c_1 \\ & \quad where \quad c_1 = (2\sigma^2 i) \\ = & ln\left(\frac{\bar{s}_i}{\bar{s}_0}\right)^2 + 2 \cdot \alpha \cdot ln(\bar{s}_0) + \alpha^2 + \\ & ln(\bar{s}_i) \cdot c_3 + c_3 \cdot log(\bar{s}_0) - c_3 \cdot ln(\bar{s}_0) \\ & \quad where \quad c_3 = c_1 - 2\alpha \\ = & \left(ln\left(\frac{\bar{s}_i}{\bar{s}_0}\right) + c_3/2\right)^2 - (c_3/2)^2 + \\ & c_3 \cdot ln(\bar{s}_0) + 2 \cdot \alpha \cdot ln(\bar{s}_0) + \alpha^2 \end{aligned} \tag{3.9}$$

Note that $\bar{s}_0$ in equation ***(3.9)*** is the recursive variable, representing the last valid estimate from the previous step.
Next, the constant from equation ***(3.9)***, specifically $c_3 \cdot log(s_0) + 2 \cdot \alpha \cdot log(s_0) + \alpha^2 - (c_3/2)^2$, will be represented as ρ_c.

$$
\begin{aligned}
&= \left(log\left(\frac{\bar{s}_i}{\bar{s}_0}\right) + c_3/2 \right)^2 + \rho_c \\
\Rightarrow &\max_{\forall s_i \in \mathbb{R}} \left(e^{\left(-\frac{(\bar{s}_i - \bar{s})^2}{2\sigma^2}\right)} \cdot e^{\left(\frac{\left(log\left(\frac{\bar{s}_i}{\bar{s}_0}\right)+c_3/2\right)^2+\rho_c}{-c_1}\right)} \right) \\
&= \max_{\forall s_i \in \mathbb{R}} \left(e^{\left(-\frac{(\bar{s}_i - \bar{s})^2}{2\sigma^2}\right)} \cdot e^{\left(\frac{\left(log\left(\frac{\bar{s}_i}{\bar{s}_0}\right)+c_3/2\right)^2}{-c_1}\right)} \right) \\
&= \max_{\forall \bar{s}_i \in \mathbb{R}} \left(e^{-\frac{(\bar{s}_{i_{meas}} - \bar{s}_i)^2}{2\cdot\sigma^2} + \frac{\left(log\left(\frac{\bar{s}_i}{\bar{s}_0}\right)+c_3/2\right)^2}{-c_1}} \right)
\end{aligned}
\tag{3.10}
$$

In equation ***(3.10)***, both e-functions have been reduced to a single e-function and the constant, ρ_c, has been eliminated. Due to the strict monotony of the e-function, it is sufficient to maximize its argument, since the point derived from the maximum argument will be the same as the point derived from maximizing the function.

$$
= \max_{\forall \bar{s}_i \in \mathbb{R}} \left(\frac{\left(log\left(\frac{\bar{s}_i}{\bar{s}_0}\right) + c_3/2\right)^2}{-c_1} - \frac{(s_i - \bar{s}_i)^2}{2 \cdot \sigma^2_{norm}} \right)
$$

To shorten the equation, four new constants, d_1 to d_4, are introduced.

$$
\begin{aligned}
d_1 &= 2\sigma^2_{norm} & d_2 &= \left(\frac{c_3}{2}\right)^2 \\
d_3 &= -2 s_i c_1 & d_4 &= c_3 d_1
\end{aligned}
$$

$$
\begin{aligned}
&= \max_{\forall \bar{s}_i \in \mathbb{R}} \left(\frac{log\left(\frac{\bar{s}_i}{s_0}\right)^2 + log\left(\frac{\bar{s}_i}{s_0}\right) c_3 + (c_3/2)^2}{-c_1} - \right. \\
&\quad \left. \frac{s_i^2 - 2 s_i \bar{s}_i + \bar{s}_i^2}{d_1} \right) \\
&= \max_{\forall \bar{s}_i \in \mathbb{R}} \left(\left(log\left(\frac{\bar{s}_i}{s_0}\right)^2 + log\left(\frac{\bar{s}_i}{s_0}\right) c_3 + d_2 \right) d_1 + \right. \\
&\quad \left. \left(s_i^2 - 2 s_i \bar{s}_i + \bar{s}_i^2\right) c_1 \right) \\
&= \max_{\forall \bar{s}_i \in \mathbb{R}} \Big(d_3 \bar{s}_i + \left(log(\bar{s}_i) - log(s_0)\right)^2 d_1 + \\
&\quad log(\bar{s}_i) d_4 + \bar{s}_i^2 c_1 \Big)
\end{aligned} \tag{3.11}
$$

In equation ***(3.11)***, the original equation, ***(3.7)***, has been reduced such that only terms dependent upon s_i remain and all static additives and constant gains have been removed. The following step calculates the derivative for equation ***(3.11)*** in order to determine the extrema points.

$$
\frac{d}{d\bar{s}_i} = d_3 + 2\bar{s}_i c_1 + \frac{2\left(ln(\frac{\bar{s}_i}{s_0})\right) d_1}{\bar{s}_i} + \frac{d_4}{\bar{s}_i} \tag{3.12}
$$

The extrema is identified by setting the derivative, $\frac{d}{d\bar{s}_i} = 0$. The logarithm prevents a closed-form solution from equation ***(3.12)*** and since a numerical solution is not desired due to its negative performance properties, the best option is to approximate the log-function. Using the Mercator formula, it is possible to calculate an accurate approximation for a given interval around an arbitrary point, $\bar{s}_i$, of the ln-function. Given that $\bar{s}_i$ represented the previous prediction of where s_i is expected, this $\bar{s}_i$ prediction should closely approximate its real position and represent an initial estimate for the starting point of the Mercator-Row. By calculating the first two elements of the Mercator-Row, leading to a quadratic function, the result is a function annealing closely to the original ln-function at an interval $\pm 50\%$ of $\bar{s}_i$.

$$
ln(\bar{s}_i) \approx ln(2)m - 1 + \frac{\bar{s}_i}{2^m} - \frac{(\frac{\bar{s}_i}{2^m} - 1)^2}{2} \tag{3.13}
$$

$$
= ln(2)m + \frac{1}{2} - \frac{1}{2}\left(\frac{s_i}{2^m} - 2\right)^2 \tag{3.14}
$$

$$
with \quad m = \frac{ln(s_0)}{ln(2)}
$$

Figure ***(Fig. 3.3)*** depicts the approximation given by equation ***(3.14)***. In figure ***(Fig. 3.3)***, the dotted line represents the original function and the continuous line represents its approximation by the quadratic polynomial function from equation ***(3.14)***. As illustrated

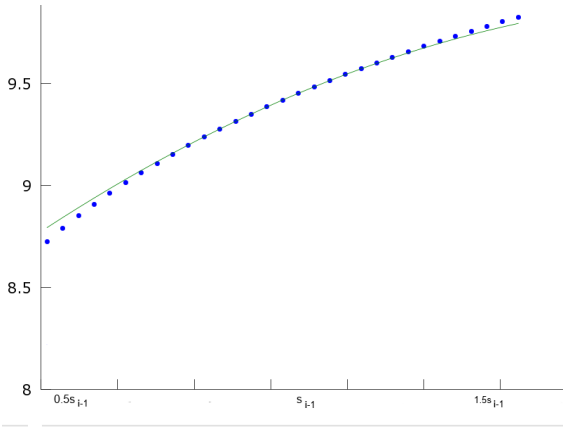

Figure 3.3: Approximation of the ln-function over the interval $\pm 50\%$ of s_{i-1}

in figure ***(Fig. 3.3)***, the approximation provides results within the interval $[0.5s_i; 1.5s_i]$. Therefore, the ln-function is replaced by the approximation of equation ***(3.14)***, resulting in equation ***(3.15)***.

$$\frac{d}{d\bar{s}_i} \approx \frac{2d_1\left(d_6 - 0.5\left(\frac{\bar{s}_i}{2^m} - 2\right)^2\right) + d_5}{\bar{s}_i} + d_3 + 2\bar{s}_i c_1 \tag{3.15}$$

$$\begin{aligned} with \quad d_5 &= d_4 - 2ln(s_0)d_1 \\ d_6 &= ln(s_0)m + 0.5 \end{aligned}$$

The terms of the approximation, ***(3.15)***, will be sorted by the factors, s_i, and all remaining constants for each factor will be represented by new constants, U_1 to U_3. This results in the final equation ***(3.16)***.

$$\frac{d}{d\bar{s}_i} \approx U_1\bar{s}_i + U_2\frac{1}{\bar{s}_i} + U_3 = 0 \tag{3.16}$$

$$\begin{aligned} where \quad U_1 &= 2c_1 - \frac{1}{2^{2m}}d_1 \\ U_2 &= 2d_6d_1 - 4d_1 + d_5 \\ U_3 &= d_3 + \frac{1}{2^{m-2}}d_1 \end{aligned}$$

Given the equation ***(3.16)***, the extrema can now be determined analytically. There are four possible solutions for equation ***(3.16)***. With respect to the constants, U_1 to U_3, only two solutions offer realistic candidates for the desired maximum. These two candidates are

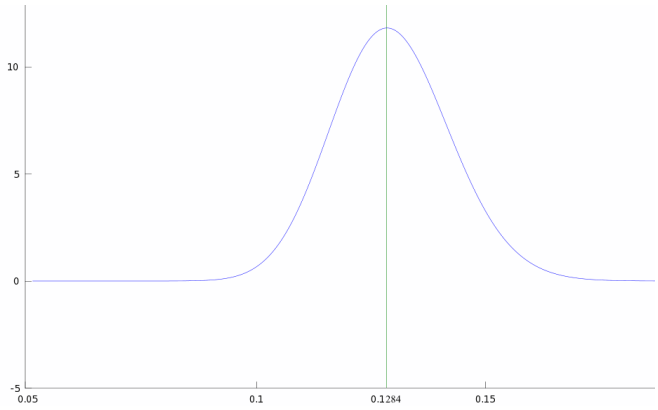

Figure 3.4: Estimated $\bar{s}_i = 0.1284$ for $s_i = 0.13$, $s_0 = 0.146$, $\mu = -0.111$, $\sigma = 0.13$ and $\hat{\sigma} = 0.02$

presented in equation ***(3.17)*** $\wedge$ ***(3.18)***.

$$l_1 \approx \frac{\sqrt{U_3^2 - 4U_1U_2} - U_3}{2U_1} \tag{3.17}$$

$$l_2 \approx -\frac{\sqrt{U_3^2 - 4U_1U_2} + U_3}{2U_1} \tag{3.18}$$

To determine whether equation ***(3.17)*** or equation ***(3.18)*** constitutes the desired maximum, the second derivative of equation ***(3.16)*** is determined. The one, l_x, where $x \in [1, 2]$ and for which ***(3.19)*** is true, constitutes the desired maximum point.

$$U_1 - U_2 \frac{1}{l_x^2} > 0 \tag{3.19}$$

Figure ***(Fig. 3.4)*** illustrates the resulting maximum within a graph overlaying the *PDF*. The density function shown in figure ***(Fig. 3.4)*** represents the original, unchanged density function. The maximum point, marked by the vertical line, is the analytical maximum determined by the approximation terms from equations ***(3.17)*** and ***(3.18)***. The developing point for the approximation, $\bar{s}_i$, was chosen to provide 20% deviation from the final maximum. The resulting approximated maximum has a deviation $< 1‰$[3]. The quality of the final approximation is strongly dependent upon distribution parameters and the expansion point, $\bar{s}_i$, of the Mercator-Row, making it difficult to ascertain approximation quality. However, while the distribution parameter are yet unknown, it is possible to choose the expansion points, $\bar{s}_i$.

3.1.4 Updating drift and variance

The final computation updates both variance, $\hat{\sigma}$, and drift, μ. The variance, $\hat{\sigma}$, quantifies the variation between the measurement estimate and the actual measurement. The drift

[3]The 'real' maximum was determined using the GNU Octave Root Finding algorithm.

factor, μ, determines the directional drift and is determined by comparing the actual estimate with that of its predecessor. Since both measurements are sensitive to noise, an averaging mechanism is introduced to decrease its influence. This mechanism is an exponential moving average, which has a higher weight on the latest predecessors instead of having a window or same weight for all previous values. Therefore, it is expected that the values will rapidly adapt to new conditions without large fluctuations between measurements. The exponential average, $\bar{X}_i$, for a measurement, X_i, at position i is given by equation ***(3.20)***.

$$\bar{X}_i = \alpha \cdot X_t + (1 - \alpha) \cdot \bar{X}_{i-1} \tag{3.20}$$

3.1.5 Implementation

Finally, all calculations require implementation into an algorithm. As mentioned previously in section *[3.1.1]*, the algorithm is divided into two steps. The first step generates a prediction[4] of the actual measurement by calculating the expectation of the GBM from equation ***(3.4)***and represents a pre-selection criterion. If the actual measurement passes this pre-selection step, the MAP-estimator will be applied to calculate a more accurate estimate.
The pre-selection is essential, given that MAP operates with the log approximation. As discussed in section *[3.1.3]*, the logarithm needs to be replaced by the Mercator-Row, which can adapt to false values as soon as the measurement becomes too distant from its expectation. Following the first step, the measure is verified to be within the acceptable range and suitable for further calculations.
The following step calculates the MAP estimate, against which the actual measurement will be once again validated. If the measurement is accepted, the variance will be updated and the algorithm restarts for the succeeding measurement based on the previously calculated MAP expectation. If at any point the actual measurement is deemed an outlier, the algorithm tests a fixed number of succeeding measurements to determine if any of them return to the expected range. This is achieved by incrementing i. If the succeeding points return to the expectation within a fixed range, all points in between will be considered as outliers and removed. Otherwise, all of these points may belong to a new segment, in which case the first outlying point will be considered as a new starting point and the algorithm restarts. Figure ***(Fig. 3.5)*** depicts the related flowchart.

[4] This prediction does not rely on the actual measurement

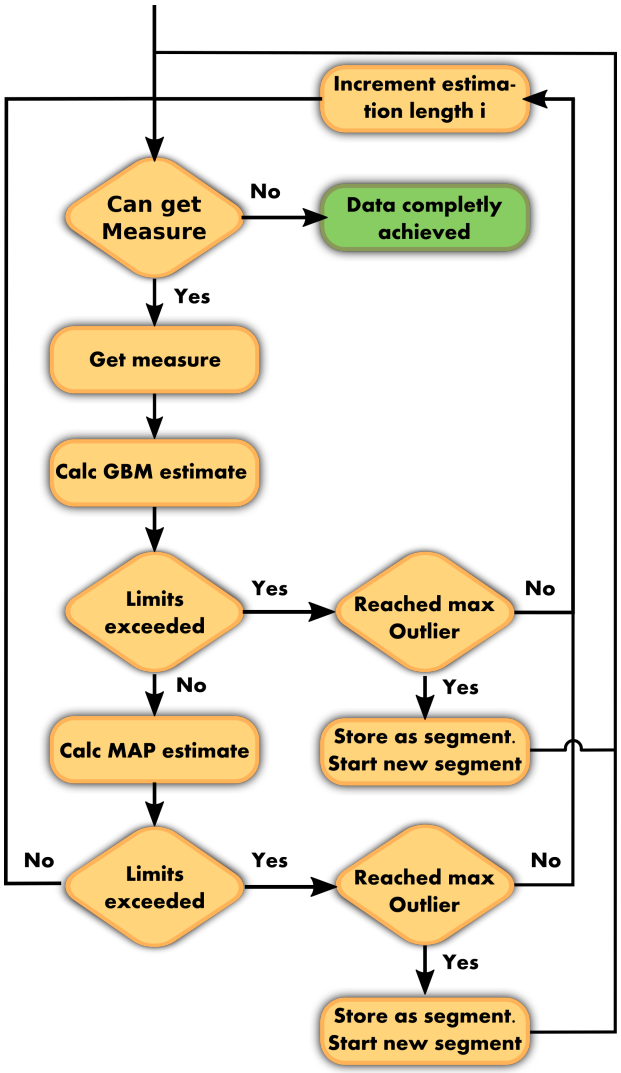

Figure 3.5: A scheme of the algorithm.

3.2 An improved method of ICP contour matching

After improving the sensor data by pure statistical means, the next logical step is to determine the desired structure within the data. Hence the following section describes a novel method to determine a predefined feature within the data.

This feature is a point within a contour segment which will be used for further measurements. The segment contour itself is provided by the sensor and can be understood as 2D point cloud. Thus, the features need to be extracted from such a point cloud. Sometimes this can be very simple. Most of the time the sensor can be fixed in a way that the point of interest is simple to extract, such as the last contour point or a top edge ***(Fig. 2.14)***. Those features can be determined almost effortlessly by algorithms, which usually run very fast. However, in some situations it might be necessary to extract measures at very complex spots of the contour, which is usually a very time-consuming task.
This feature is a point within a contour segment which will be used for further measurements. The segment contour itself is provided by the sensor and can be understood as 2D point cloud. Thus, the features need to be extracted from such a point cloud. Sometimes this can be very simple. Most of the time the sensor can be fixed in a way that the point of interest is simple to extract, such as the last contour point or a top edge []. Those features can be determined almost effortlessly by algorithms, which usually run very fast. However, in some situations it might be necessary to extract measures at very complex spots of the contour, which is usually a very time-consuming task.

In the publication[48] we proposed a mechanism that determines contour spots of arbitrary shapes very fast. To achieve this, we offered a matching algorithm that integrated a two-step approach. First, the points have been reduced in order to perform as fast pre-position on the reduced point set. After having a pre-position, a final positioning step ensures the accurate fitting.

The proposed algorithm is able to match an arbitrary predefined contour pattern into the corresponding position within the measurement. Thus, if there is a measurement spot related to the pattern, this spot will be rediscovered within scans of the contour. Since the last decades, point cloud matching has been a very active field of research in machine learning. One of the early papers on this topic was [49] describing a method for *"representation-independent .. registration of 3-D shapes"*[5], the so-called iterative closest point algorithm (ICP). Many variations of this algorithm have been developed, each resulting in advantages, but also some disadvantages. The original formulation of the ICP depends strongly on the choice of good start values, therefore the improvement of robustness of the ordinary ICP is and has always been one of the main topics for further research. Since the lack of robustness is a direct consequence of the hard point-to-point correspondences used in the ordinary ICP, the most promising approaches use a soft correspondence for the point matches. Today there are two common specializations of the ordinary ICP-the Softassign-ICP *(or SA-ICP)* by Gold et al. [50] and the EM-ICP by Granger et al. [51]. Both use the above-mentioned soft correspondence for the point relationship. The SA-ICP uses a *softmax*-function, usually as an

[5] p.239 [49]

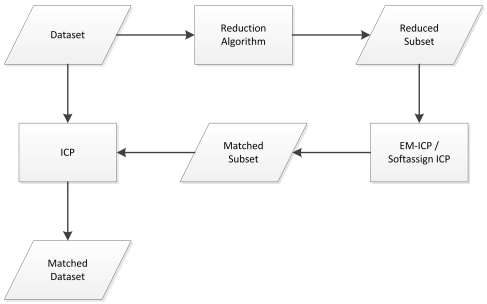

Figure 3.6: Sketch of the proposed algorithm

activation function in connection with neuronal networks, to archive the soft correspondence in between the scene- and model-points. The EM-ICP on the other hand uses the *matching probability* as a weight for correspondences. Both approaches lead to a significantly smaller number of local minima and therefore to higher robustness with regard to the start values. We chose the EM-ICP, since the computation of point correspondences for the SA-ICP seems to be an adverse, due to the *Shinkhorn* iteration [52]. But one could easily replace the EM-ICP with the SA-ICP for the given description, if desired.

To provide a robust and also fast-matching algorithm, the computationally intensive EM-ICP is only used as a means of prepositioning a reduced point set. This results in an adequate set of start values for use with an ordinary ICP on the full set, which will lead to a high level of convergence. The aforementioned reduced point set is generated by using Ramer's curve approximation algorithm [53], which reduces the original point set to a number of marginal, but *significant* points. That algorithm and its implementation will be introduced in the first section.

After that, we give a short outline of the EM-ICP and its implementation. Finally, we present the last step of the point cloud matching algorithm, the ordinary ICP implemented in combination with kD-tree-based neighbor search [54].The idea of the proposed is sketched in Fig. 3.6.

3.2.1 Point reduction

Due to the soft correspondence property of the EM-ICP, the cost of calculation is significantly higher than that of an ordinary ICP. This is due to the costs of calculation of the correspondences itself on the one hand and the calculation of the transformations between the matching points on the other hand, due to the fact that one model point can match various scene points. Therefore, Granger et al. [51] introduced a decimation scheme, called *"the greedy sphere decimation"*: only those points are taken into account that are within a sphere centered on the barycenter of its containing points and has a radius of $\alpha \cdot \sigma$ (where $\alpha \in \mathbb{R}$ is a constant and σ is the standard deviation over the actual estimation of the variance). This makes sense since points beyond this sphere would receive an insignificant weight and therefore would not be of interest. Since σ shrinks during the iteration process we might still use too many points during the first iterations.

To be more efficient, it would be advantageous to reduce the overall scene measurements in such a way that the loss of information is minimized on those parts of the point cloud which are rich in information and sparse the point cloud where the measurements are unserviceable. That is to say, scene measurements in regions of curvature should be left untouched because those regions are important for the matching, whereas linear or plane segments are less important and we can accept them to be sparse. This leads to a polygonal curve representation in 2D space for 2D data, or to a polygonal surface triangulation for 3D data, where the distance between the representation and the closest measurement never exceeds a fixed-threshold value. Since this algorithm was meant to fit contours, this thesis will discuss the 2D case [6].

Such a curve representation has been introduced by Ramer [53][7]. Ramer tried to reduce the curve to *"polygons with few edges but which still retain the significant features of the curves they represent"*[8]. This algorithm expects an ordered set of point measurements $\mathfrak{M}_s$ which then will be reduced to a subset $\mathfrak{M}_s^{'} \subseteq \mathfrak{M}_s$ gathering those points $\underline{p}' \in \mathfrak{M}_s^{'}$ which satisfy the condition ***(3.21)***.

$$\max_{\forall \underline{p}_i} \left(\perp \left(\underline{p}_i, \left\langle \underline{p}'_{k-1}, \underline{p}'_k \right\rangle \right) \right) \leq \alpha \tag{3.21}$$

Where $\underline{p}_i \in \left\{ \underline{p} \,\middle|\, \underline{p}'_{k-1} \leq \underline{p} < \underline{p}'_k, \underline{p} \in \mathfrak{M}_s, \underline{p}' \in \mathfrak{M}_s^{'} \right\}$ $\underline{p}'_{k-1}, \underline{p}'_k \in \mathfrak{M}_s^{'}$. The functional $\langle \cdot, \cdot \rangle$ denotes the linear interpolation between the two neighboring points and the function $\perp (\cdot, \cdot)$ calculates the perpendicular distance in between a point $\underline{p}$ and the interpolation $\langle \cdot, \cdot \rangle$.

To generate the desired set $\mathfrak{M}_s^{'}$, two points need to be chosen as initial values. For contours that are not closed, these points are the left- and right-most points $\underline{p}_1$ and $\underline{p}_{|\mathfrak{M}_s|}$. The next step is to search the most distant contour point $\underline{p}_i$ from the linear interpolation in between those two starting points. If this point exceeds condition ***(3.21)***, the contour will be divided at p_i from the linear interpolation in between those two starting points. This will be repeated until condition ***(3.21)*** is not exceeded in any subsegment. By recursively breaking the given sets down to subsets, this algorithm can be classified as a *divide-and-conquer* algorithm which leads to an average time complexity of $O(n \, log \, n)$, given a worst case complexity of $O\left(n^2\right)$ but only under the circumstances that α has been chosen inappropriately. Therefore, the Ramer algorithm is a very efficient technique to reduce the number of points drastically[9], especially within plane or straight segments of less importance for the EM-Prepositioning. A graphical example of the Ramer-Algorithm is shown in Fig. 3.7.

[6] Although the discussed techniques could be extended to process 3D data

[7] Which was also independently developed by David Douglas and Thomas Peucker [55] and therefore is sometimes referred to as the *"Douglas Peucker algorithm"*

[8] p. 245 [53]

[9] Unfourtunatelly an exact shrinkage cannot be determined due to the fact, that the shrinkage depends on the contour's shape and the chosen α.

Algorithm 1 Reduce_Meas_Points$\left(\hat{\mathfrak{M}}\right)$

Require: $\hat{\mathfrak{M}} \subseteq \mathfrak{M}_s$
Ensure: $\mathfrak{M}^{'} := \{\underline{p} \,|\underline{p}$ satisfies condition ***(3.21)*** $\}$
  $p_x \leftarrow \varnothing$
  $maxDist \leftarrow 0$
  $\ell \leftarrow \left\langle \underline{p}_1, \underline{p}_{|\hat{\mathfrak{M}}|} \right\rangle$
  for $i = 1$ to $\left|\hat{\mathfrak{M}}\right|$ **do**
    if $\perp \left(\underline{p}_i, \ell\right) > maxDist$ **then**
      $maxDist \leftarrow \perp \left(\underline{p}_i, \ell\right)$
      $\underline{p}_x \leftarrow \underline{p}_i$
    end if
  end for
  if $maxDist > \alpha$ **then**
    $\hat{\mathfrak{M}}_1 := \left\{\underline{p} \,\middle|\, \underline{p}_1 \leq \underline{p} < \underline{p}_x\right\}$
    $\hat{\mathfrak{M}}_2 := \left\{\underline{p} \,\middle|\, \underline{p}_x < \underline{p} \leq \underline{p}_{|\hat{\mathfrak{M}}|}\right\}$
    $\mathfrak{M}_1^{'} \leftarrow$ **Recursive call** *Reduce_Meas_Points* $\left(\hat{\mathfrak{M}}_1\right)$
    $\mathfrak{M}_2^{'} \leftarrow$ **Recursive call** *Reduce_Meas_Points* $\left(\hat{\mathfrak{M}}_2\right)$
    return $\mathfrak{M}_1^{'} \cup \mathfrak{M}_2^{'} \cup \left\{\underline{p}_x\right\}$
  end if
  return $\varnothing$

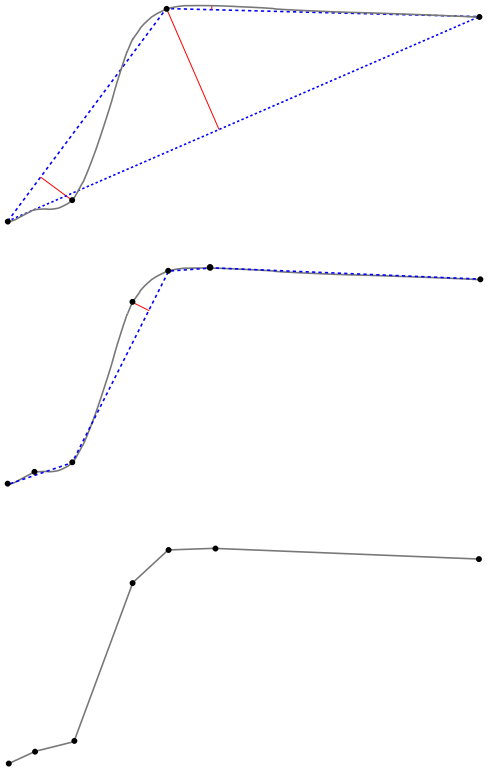

Figure 3.7: Example of the Ramer-Algorithm

3.2.2 The EM-ICP prepositioning

After the amount of points was reduced in the first step, the next step is the prepositioning of the reduced sets. For this step the aforementioned EM-ICP is the instrument of choice. This is not only due to its strong convergence property, but also because the EM-ICP is a fully stochastic approach which models the transformation $\boldsymbol{T}$ in between the model and scene points, as well as the matches in between the point sets $\boldsymbol{A}$ as a stochastic variable, based on the EM[10] algorithm [56]. Since there are no fixed point matches, it was valid to reduce the number of points as described in section 3.2.1. Or, in other words, reducing the point as described in section 3.2.1 is only possible because there is a *soft* correspondence between the model and scene points.

We assume the set of scene points to be $\underline{s_i} \in \mathfrak{S}$ and the model points to be $\underline{m_i} \in \mathfrak{M}$. The first step done by Granger et al. [51] is to model the probability of a scene point, given the model points and a transformation $\boldsymbol{T}$. As soon as the scene points are expected to be

[10] Expectation maximization algorithm

Gaussian distributed with zero mean and Σ variance, this leads to equation ***(3.22)***.

$$p\left(\underline{s}_i \left| \underline{m}_j, \boldsymbol{T}\right.\right) = k^{-1} e^{-0.5 \cdot f_m\left(\boldsymbol{T} \cdot \underline{s}_i, \underline{m}_j\right)} \tag{3.22}$$

Where k is the norm factor given by equation ***(3.23)*** and the function $f_m\left(\boldsymbol{T} \cdot \underline{s}_i, \underline{m}_j\right)$ calculates the Mahalanobis distance in between a model and a scene point with respect to the transformation, given by equation ***(3.24)***.

$$k = (2\pi)^{\frac{p}{2}} \left|\boldsymbol{\Sigma}\right|^{\frac{1}{2}} \tag{3.23}$$

$$f_m(\underline{x}_1, \underline{x}_2) = (\underline{x}_1 - \underline{x}_2)^{tr} \boldsymbol{\Sigma}^{-1} (\underline{x}_1 - \underline{x}_2) \tag{3.24}$$

The probability given in equation ***(3.22)*** works only for corresponding points. This correspondence will be managed by a matching matrix $\boldsymbol{A}$. In the simplest case $\boldsymbol{A}$ can be a binary matrix which is $\boldsymbol{A}_{ij} = 1$ if a scene point $\underline{s}_i$ corresponds with model a point $\underline{m}_j$, or 0 if they do not correspond. Since these correspondences are independent, the probability of $p(\boldsymbol{A})$ is given by equation ***(3.25)***.

$$p(\boldsymbol{A}) := \prod_{i}^{|\mathfrak{S}|} \prod_{j}^{|\mathfrak{M}|} \left(p(\boldsymbol{A}_{ij} = 1)\right)^{\boldsymbol{A}_{ij}} \tag{3.25}$$

This results in a maximum probability $p(\boldsymbol{A}) = 1$ for a binary matching matrix, where we have a unique matching for each point. But, since we want a soft correspondence, we allow each matching expectation to be $\boldsymbol{E}(\boldsymbol{A}_{ij}) = p(\boldsymbol{A}_{ij} = 1) \in [0, 1]$ as a convex combination over all model points (equation ***(3.26)***).

$$\sum_{j}^{|\mathfrak{M}|} p(\boldsymbol{A}_{ij} = 1) = \sum_{j}^{|\mathfrak{M}|} \boldsymbol{E}(\boldsymbol{A}_{ij}) = 1 \tag{3.26}$$

With the given probability of $\boldsymbol{A}$ in equation ***(3.25)*** and the matching probability of equation ***(3.22)***, we can now rewrite the conditioned probability for a single scene point $\underline{s}_i$ with respect to the model, the transformation, and the correspondence matrix, as in equation ***(3.27)***.

$$\begin{aligned} p\left(\mathfrak{S} \left| \boldsymbol{A}, \boldsymbol{T}, \mathfrak{M}\right.\right) &= \prod_{i}^{|\mathfrak{S}|} p\left(\underline{s}_i \left| \boldsymbol{A}, \boldsymbol{T}, \mathfrak{M}\right.\right) \\ &= \prod_{i}^{|\mathfrak{S}|} \prod_{j}^{|\mathfrak{M}|} \left(p\left(\underline{s}_i \left| \boldsymbol{T}, \underline{m}_j\right.\right)\right)^{\boldsymbol{A}_{ij}} \end{aligned} \tag{3.27}$$

Now the a-priori probability for $\boldsymbol{A}$ is chosen to be uniformly distributed $p(\boldsymbol{A}_{ij} = 1) = \overline{\pi_{ij}} = |\mathfrak{M}|^{-1}$ as stated in [51]. This leads to the scene matching likelihood of equation ***(3.28)*** and

the single scene point probability of equation ***(3.29)***.

$$\begin{aligned} p\left(\mathfrak{S}, \boldsymbol{A} \left| \boldsymbol{T}, \mathfrak{M} \right.\right) &= p\left(\mathfrak{S} \left| \boldsymbol{A}, \boldsymbol{T}, \mathfrak{M} \right.\right) \cdot p\left(\boldsymbol{A} \left| \boldsymbol{T}, \mathfrak{M} \right.\right) \\ &= \prod_{i}^{|\mathfrak{S}|} \prod_{j}^{|\mathfrak{M}|} \left(\overline{\pi_{ij}} \cdot p\left(\underline{s}_i \left| \boldsymbol{T}, \underline{m}_j \right.\right)\right)^{\boldsymbol{A}_{ij}} \end{aligned} \tag{3.28}$$

$$p\left(\underline{s}_i \left| \mathfrak{M}, \boldsymbol{T} \right.\right) = \sum_{j}^{|\mathfrak{M}|} \overline{\pi_{ij}} \cdot p\left(\underline{s}_i \left| \underline{m}_j, \boldsymbol{T} \right.\right) \tag{3.29}$$

From now on, given the scene, the model and the transformation, the probability of $\boldsymbol{A}$ can be derived by Bayes' theorem, as stated in equation ***(3.30)***.

$$p\left(\boldsymbol{A} \left| \boldsymbol{T}, \mathfrak{S}, \mathfrak{M} \right.\right) = \frac{p\left(\mathfrak{S}, \boldsymbol{A} \left| \boldsymbol{T}, \mathfrak{M} \right.\right)}{p\left(\mathfrak{S} \left| \boldsymbol{T}, \mathfrak{M} \right.\right)} = \prod_{i}^{|\mathfrak{S}|} \prod_{j}^{|\mathfrak{M}|} \left(\frac{\overline{\pi_{ij}} \cdot p\left(\underline{s}_i \left| \boldsymbol{T}, \underline{m}_j \right.\right)}{\sum_{k}^{|\mathfrak{M}|} \overline{\pi_{ik}} \cdot p\left(\underline{s}_i \left| \boldsymbol{T}, \underline{m}_k \right.\right)} \right)^{\boldsymbol{A}_{ij}} \tag{3.30}$$

For a single correspondence probability, this leads to equation ***(3.31)*** with respect to equation ***(3.25)***.

$$\boldsymbol{E}(\boldsymbol{A}_{ij}) = \frac{\overline{\pi_{ij}} \cdot e^{-0.5 \cdot f_m\left(\boldsymbol{T} \cdot \underline{s}_i, \underline{m}_j\right)}}{\sum_{k}^{|\mathfrak{M}|} \overline{\pi_{ik}} \cdot e^{-0.5 \cdot f_m\left(\boldsymbol{T} \cdot \underline{s}_i, \underline{m}_k\right)}} \tag{3.31}$$

Equation ***(3.25)*** represents the *"E-Step"* of the EM-Algorithm, as soon as it denotes the estimation of the matching matrix. For the maximization step *("M-Step")* we can simply use the full scene probability given in ***(3.28)*** and build the log likelihood, equation ***(3.32)***.

$$-\boldsymbol{E}(log\, p\left(\mathfrak{S}, \boldsymbol{A} \left| \boldsymbol{T}, \mathfrak{M} \right.\right)) = -\sum_{ij} \boldsymbol{A}_{ij}\ \log\ \overline{\pi_{ij}} p\left(\underline{s}_i \left| \underline{m}_j, \boldsymbol{T} \right.\right) \tag{3.32}$$

Since the Mahalanobis distance function f_m is reduced to equation ***(3.33)*** for isotropic and uncorrelated noise

$$f_m(\underline{x}_1, \underline{x}_2) = \left\| \underline{x}_1 - \underline{x}_2 \right\|^2 \sigma^{-2} \tag{3.33}$$

and the constants do not affect the minimization, we can rewrite equation ***(3.32)*** to equation ***(3.34)***, which enables us to find the optimal transformation for the given *configuration*.

$$\begin{aligned} &\min_{\forall \boldsymbol{T}} \left(-log\, \mathcal{L}\left(\boldsymbol{T} \left| \sigma^2, \boldsymbol{A}, \mathfrak{S}, \mathfrak{M} \right.\right)\right) \\ = &\min_{\forall \boldsymbol{T}} \left(\sum_{ij} \boldsymbol{A}_{ij} \left\| \underline{s}_i - \boldsymbol{T} \underline{m}_j \right\|^2 \sigma^{-2} \right) \end{aligned} \tag{3.34}$$

The last missing variable is σ^2, which is the variance estimate, but since Granger states in [57] that an estimation of σ^2 tends to be unstable because of its rapid shrinkage, a constant σ^2 is used, which will be multiplied by an annealing coefficient each iteration step.
These results can now be formulated as an algorithm 2. Since during the estimation step the calculation of $\boldsymbol{A}_{ij}$ is independent for each i and j, it can be computed in parallel. For a discussion of possible optimations see Tamaki et al. [52], where details about the implementation of an EM-ICP algorithm in CUDA and OpenMP are provided.

Algorithm 2 Calculate_EM-ICP($\mathfrak{S}, \mathfrak{M}$)

Require: $\mathfrak{S}, \mathfrak{M}$
Ensure: Max. likelihood estimate of $\boldsymbol{T}$
  $\boldsymbol{T}_{new} \leftarrow$ initial $\boldsymbol{T}$
  repeat
    for $i = 0$ to $|\mathfrak{S}|$ **do**
      for $j = 0$ to $|\mathfrak{M}|$ **do**
        $\boldsymbol{A}^*_{ij} \leftarrow \frac{\overline{\pi_{ij}} \cdot e^{-0.5 \cdot f_m\left(\boldsymbol{T}_{new} \cdot \underline{s}_i, \underline{m}_j\right)}}{\sum_k^{|\mathfrak{M}|} \overline{\pi_{ik}} \cdot e^{-0.5 \cdot f_m\left(\boldsymbol{T}_{new} \cdot \underline{s}_i, \underline{m}_k\right)}}$
      end for
    end for
    $\boldsymbol{T}_{old} \leftarrow \boldsymbol{T}_{new}$
    $\boldsymbol{T}_{new} \leftarrow$ FindMinArg $\left(\sum_{ij} \boldsymbol{A}^*_{ij} \left\|\underline{s}_i - \boldsymbol{T}\underline{m}_j\right\|^2\right)$
    $\mathfrak{M} \leftarrow$ TransformModel($\boldsymbol{T}_{new}$)
  until $\|\boldsymbol{T}_{new} - \boldsymbol{T}_{new}\| < \epsilon$

3.2.3 Finalizing with the ordinary ICP

The ordinary ICP is used as the last step to finalize the matching process. From here on, one needs to continue with the original point set, not the reduced version used in section 3.2.2. Since a good guess of the contour position is known due to the EM-ICP, we can expect the ICP to converge in only a few iterations. The neighbor search is implemented as a kD-Tree search, since this is a very intuitive and efficient improvement that was already stated in the original paper [49] as an outlook[11]. kD-trees are virtually dividing the k-dimensional space into subspaces, which are organized by a tree structure and can therefore be searched very quickly *($O(log\, n)$ for the search of the nearest neighbor point)*. Though the neighbor search is very fast, the construction of a kD-Tree is about $O(k\, n\, log\, n)$. Therefore, it makes sense to build it in parallel during the prepositioning. Since kD-trees are very common in the field of image processing and can be considered to be a standard technique, the kD-trees algorithm shall not be discussed in this work[12].

The ICP algorithm works by searching the nearest neighbor scene point $\underline{s}_i$ for each model point $\underline{m}_j$, with respect to the Euclidean distance. Those two points are assumed to be matches. This search can be accelerated with the kD-trees. Now one can consider a matching matrix in a similar way as in section 3.2.2, but in this case as a binary matrix denoting 1 for a match and 0 otherwise (equation ***(3.35)***).

$$\boldsymbol{A}_{ij} := \begin{cases} 1 & if \;\; \|m_j - s_i\| = \min\limits_{\forall m_k \in \mathfrak{M}} \|m_k - s_i\| \\ 0 & otherwise \end{cases} \tag{3.35}$$

And, since the likelihood of the scene given in equation ***(3.27)*** in section 3.2.2 is a very general assumption, one can formulate the likelihood for the transformation given the fixed

[11] p. 254 "VIII. FUTURE DIRECTIONS" [49]
[12] Further readings can be found under [58] and [59]

binary matching matrix of equation ***(3.35)*** similarly as denoted in equation ***(3.36)***.

$$\mathcal{L}\left(\boldsymbol{T}\,|\boldsymbol{A},\mathfrak{S},\mathfrak{M}\right)=p\left(\mathfrak{S}\,|\boldsymbol{A},\boldsymbol{T},\mathfrak{M}\right) \tag{3.36}$$

From there on, the next steps are straight forward since one needs to rewrite it as a negative log-likelihood function. This results in equation ***(3.37)***.

$$\begin{aligned} & -\log\left(\mathcal{L}\left(\boldsymbol{T}\,|\boldsymbol{A},\mathfrak{S},\mathfrak{M}\right)\right) \\ = & -\log\left(\prod_{i}^{|\mathfrak{S}|}\prod_{j}^{|\mathfrak{M}|}\left(p\left(\underline{s}_i\,|\boldsymbol{T},\underline{m}_j\right)\right)^{\boldsymbol{A}_{ij}}\right) \\ = & \sum_{ij}^{|\mathfrak{S}||\mathfrak{M}|}\left[\boldsymbol{A}_{ij}\left\|\underline{s}_i-\boldsymbol{T}\underline{m}_j\right\|^2\sigma^{-2}\right] \end{aligned} \tag{3.37}$$

In equation ***(3.37)*** Σ^{-1} has already been replaced with σ^{-2} which denotes the isotropic and uncorrelated noise assumption, similar as in section 3.2.2. To find the optimal match, we assume σ to be one, as in the standard normal distribution, and search for a $\boldsymbol{T}$ which minimizes equation ***(3.38)***.

$$\min_{\forall\boldsymbol{T}}\sum_{ij}^{|\mathfrak{S}||\mathfrak{M}|}\left[\boldsymbol{A}_{ij}\left\|\underline{s}_i-\boldsymbol{T}\underline{m}_j\right\|^2\right] \tag{3.38}$$

This needs to be repeated iteratively until we converge into our local minimum. This leads to the formulation of algorithm 3.

Algorithm 3 Ordinary_ICP($\mathfrak{S},\mathfrak{M}$)

Require: $\mathfrak{S},\mathfrak{M}$
Ensure: Max. likelihood estimate of $\boldsymbol{T}$
  $\boldsymbol{T}_{new}\leftarrow inf$
  repeat
    $\boldsymbol{A}\leftarrow\boldsymbol{A}\cdot 0$
    for $i=0$ to $|\mathfrak{S}|$ **do**
      $j\leftarrow$ FindNNeighborIdx($\underline{s}_i$) {kD-Search}
      $\boldsymbol{A}^*_{ij}\leftarrow 1$
    end for
    $\boldsymbol{T}_{old}\leftarrow\boldsymbol{T}_{new}$
    $\boldsymbol{T}_{new}\leftarrow$ FindMinArg $\left(\sum_{ij}\boldsymbol{A}^*_{ij}\left\|\underline{s}_i-\boldsymbol{T}\underline{m}_j\right\|^2\sigma^{-2}\right)$
    $\mathfrak{M}\leftarrow$ TransformModel($\boldsymbol{T}_{new}$)
  until $\|\boldsymbol{T}_{new}-\boldsymbol{T}_{new}\|<\epsilon$

3.2.4 Finding the minimum Transformation

In both algorithms, algorithm 2 and algorithm 3 in section 3.2.2, one can find a function named *"FindMinArg"*. This method is meant to search the transformation that minimizes

the criterion passed as the argument. The minimizing $\boldsymbol{T}$ can be found in a closed form solution for a given criterion in $\mathbb{R}^2$ as well as in $\mathbb{R}^3$. A common solution for $\mathbb{R}^3$ based on quaternions is given by Horn [60]. But, since this algorithm is meant to work in $\mathbb{R}^2$ with the purpose of contour matching, we are going to illustrate how to find the minimizing $\boldsymbol{T}$ in $\mathbb{R}^2$, which is a linear least square estimator.
Equation ***(3.39)*** illustrates the related linear model,

$$\boldsymbol{A} \cdot \underline{\beta} = \underline{b} \tag{3.39}$$

$$\Rightarrow \quad \underline{\beta} = \left(\boldsymbol{A}^T \boldsymbol{A}\right)^{-1} \boldsymbol{A}^T \underline{b} \tag{3.40}$$

where $\boldsymbol{A}$ is the design matrix, the vector $\underline{b}$ is the resulting vector and the vector $\underline{\beta}$ is the vector of unknowns. Then the best linear unbiased estimation of $\underline{\beta}$ is denoted to be as in equation ***(3.40)***.
If one choses $\boldsymbol{A}$ to be as in equation ***(3.41)*** and $\underline{b}$ as in equation ***(3.42)***, the best linear unbiased estimation of $\boldsymbol{T}$ is given as in equation ***(3.43)***.

$$\boldsymbol{A} = \begin{pmatrix} x_1 & y_1 & 1 & 0 \\ x_1 & y_1 & 0 & 1 \\ x_2 & y_2 & 1 & 0 \\ x_2 & y_2 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & 1 & 0 \\ x_n & y_n & 0 & 1 \end{pmatrix} \tag{3.41}$$

$$\underline{b} = \begin{pmatrix} x_1^* \\ y_1^* \\ x_2^* \\ y_2^* \\ \vdots \\ x_n^* \\ x_n^* \end{pmatrix} \tag{3.42}$$

$$\boldsymbol{T} = \begin{pmatrix} \beta_1 & \beta_2 & \beta_3 \\ -\beta_2 & \beta_1 & \beta_4 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.43}$$

Where the resulting matrix $\boldsymbol{T}$ represents a homogeneous transformation matrix in $\mathbb{R}^2$, in which the squared 2×2 sub-matrix $t_{1,1}$ to $t_{2,2}$ represents the rotation and the sub-vector $(t_{1,3}, t_{2,3})^T$ represents the shift of the contour.

3.3 Position recognition with sparse data

The last two sections concentrate on the data provided by the sensor. The described methods to improve noisy data (section *[3.1]*) as well as methods to determine pre-learned features within the sensor data (section *[3.2]*). Hence, the next logical step is to determine which information from those extracted features is applicable for the robot in order to optimize its motion path. The method described in this chapter is going back on a scheme which was supplied by Philipp Roebrock in a technical paper for Polytechnical University in Timisoara in his Ph.D. program[13]. His basic scheme of a path-correction algorithm has been picked up, extended, and reformulated within this thesis in a way that serves the needs of industrial path correction. The algorithm has been implemented within the VMT BK path correction system and serves in many sealing applications worldwide, today. Therefore, the following section will introduce a novel method of path correction, which can combine multiple features extracted from the sensor data to a fully qualified correction vector.

Path correction means correcting an application path of the robot with single or multiple corrections. There are many ways to achieve corrections for the robot, such as those mentioned in [61],[62] and [19]. Most approaches are online corrections, which act like a control loop in which the recent measurement has a direct effect on the robot path, called point-to-point correction.

Figure ***(Fig. 3.8)*** depicts a typical application spot for a path correction system, which lies at the back side of a door. The bead marked in red is the sealing bead that covers the gap between the crimped edge, which is connecting the outer and inner metal part of the door. This type of bead will stay visible for the customer and might only be partly covered by

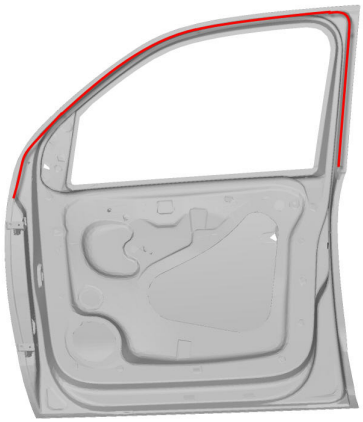

Figure 3.8: Depicts a typical sealing bead, marked in red, on the backside of a door.

plastic lids. Most of it might remain visible and the customer will be able to see it whenever he is entering the car. Since this bead also runs parallel to the door edge, a human observer

[13] Unfortunately this technical paper has never been published

will have a straight reference edge. As described in section *[1.2]*, this setup will allow the human observer to see even the slightest inaccuracies and deviations. Hence, one can easily imagine that there have to be visible defects within a point-to-point correction in cases of :

- bigger but necessary adjustments $(> 0.2mm)$, causing tiny saw tooth shapes
- erroneous measures, since the correction is point by point, foreign particles or noise

This can lead to huge quality issues. It is, however, more desirable, to have a smooth alinement along the reference edge, and combine multiple measures to a single correction to be protected from single erroneous measurements. Thus, one of the key contributions of this work is a novel algorithm, introduced in *"An over-determined path correction algorithm for sparse dimensional measurements"*[4], which combines multiple measures to a single 6 dimensional correction vector. The proposed algorithm can collect the information in a so-called measurement run, combining this information with correction vectors, and transmitting them to the robot during application. It is also possible to define multiple correction spots with separate corrections, in order to respect the lower rigidity of most of the parts. For such correction spots, multiple measurement positions will be combined to a correction region. Since the robot switches the correction by activating frames when entering a new region, its corrections will be smooth and no sawtooth-like jumps will be visible on the door contours. Figure ***(Fig. 3.9)*** shows a typical configuration for a car door. The sensors indicate the

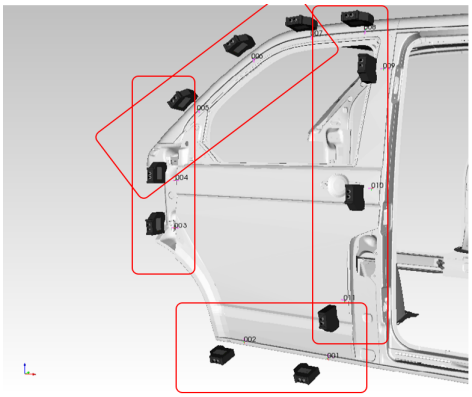

Figure 3.9: Sensors indicate the measurement positions and the red boxed indicate the measures which have been combined to a correction.

spots where the robot triggered the measures. All measurement spots gathered in a red box are combined to a single correction region. It follows that, since multiple measures are combined for a single correction, the resulting quality is higher and less noise affected due to over determination. And, finally, there is only one correction on each edge. To be able to position the sensor at the predefined measurement positions, as indicated in figure ***(Fig. 3.9)***, the sensor will be mounted on the robot's wrist as shown in ***(Fig. 3.10)***. The robot will run two specially prepared programs, which must be generated only once when the robot is set up. One is called the measuring program and it contains all the predefined measuring

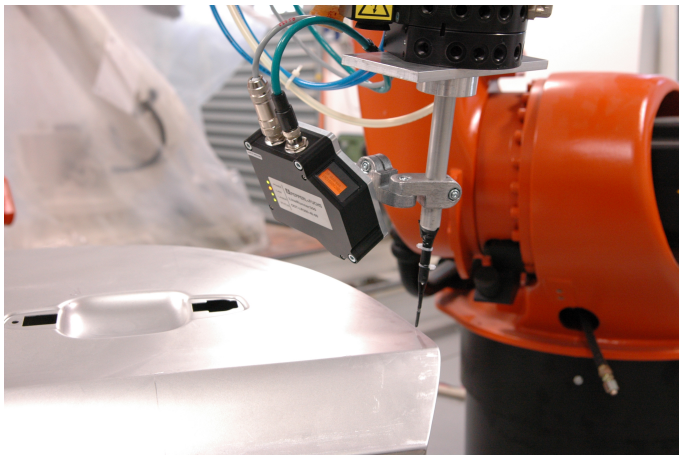

Figure 3.10: Path correction demo system with wrist mounted sensor.

spots, such as those in ***(Fig. 3.9)***. At each measurement spot, a specially prepared trigger command will be called, which is provided by the path-correction driver package[14] installed on the robot. Whenever the sensor is triggered, it will record an image and send it to the main system, where the correction vectors will be calculated. After calculating the corrections the main system will send them back to the robot, where the driver modules will receive them. Finally in the application run, the driver module will turn the corrections at special, predefined correction points, which are called application points, on[15].
Since the robot is not stopping while it triggers the measurements during the measurement program, the exact measurement position will be undefined within a small range around the desired spot. This is called the trigger jitter. This jitter is caused by various reasons. The major delay originates from the time span between the point where the driver module on the robot signaled the trigger to the point where it arrives at the sensor. This is strongly connected to the way the sensor hardware trigger is connected. If the connection is via a filedbus system, it might already be within a milliseconds range and, depending on the motion speed of the robot during measurement, it might already cause deviations of about a few millimeters. But also, the sensor, which continuously records images, has just delivered the next available image after a trigger event. So the delay is undefined within the image acquisition time on the sensor as well.Figure ***(Fig. 3.11)*** illustrates the trigger latency problem for a single measurement spot. Since this jitter effect is of significant strength and cannot be ignored within the position calculations, the following proposed algorithm defines a special non-considered degree of freedom, which needs to be aligned in the direction of robot motion. Because the laser line scanner only provides measurements within the X/Z-Plane, it is desirable to arrange the measurement program in such a way that the non-measurable degree of freedom within the sensor is aligned with the direction of robot motion at the

[14] The path correction system has been developed in cooperation with VMT Vision Machine Technic Bildverarbeitungssysteme GmbH, who supplied the robot driver package

[15] "turning on" in this case means, that the base/workobject of the robot will be shifted by the correction vector

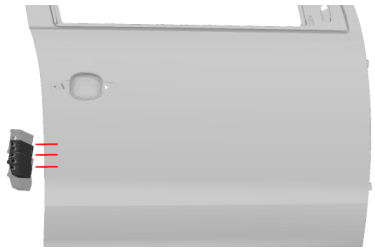

Figure 3.11: Sensor triggered in motion, so trigger position might vary.

trigger positions. In this case, the full X/Z-Information of the sensor will be used for position calculation. If for some reason it is not possible to align the measurement plane orthogonal to the motion direction, which might be at spots with limited reachability, then the weight vector [16] can be used to balance each component in a way that the weights cancel the skewness. To achieve such a parallel configuration within the trigger range at the measurement spot, the programs for the robot are prepared in a way that will ensure an entry point within about 20mm before the trigger position and also an exit point about 20mm after the trigger position. Those positions are set up so that at each of those points the sensor image is equal. The robot motion type is linear and the Y-direction of the sensor is aligned to the motion direction. The stability of the setup can be verified by multiple repeated measurements, which should all result in a zero shift.

3.3.1 Calculation of the correction vector

Combining all measurements to a fully qualified corrections vector[17] is challenging since the sensor measurements are sparse. "Sparse" means that the sensor only provides two dimensional information in X an Z. As described in detail in section *[2.4.2]*, this sparsity causes multiple problems and therefore such values cannot be used for correction directly.One is also facing the problem described above, which is that the measurement position is some how uncertain due to the trigger latency. Also, since the desired system is a measurement system, we can expect the part to move as well. In other words, the position where the measurement has been made is completely uncertain and it is essential for the proposed algorithm to consider these circumstances in order to achieve stable results. Image ***(Fig. 3.12)*** depicts this type of change in circumstances. Since the original measurement position, from sub image A is preserved within image B, one is able to see the deviations $(\Delta x, \Delta z)$ that will be measured by the sensor in situation B. It is also clear, that the values $(\Delta x, \Delta z)$ hardly provide any relation to the original spot measured in A, which is marked by the frame (x', y', z') in image B.
To understand how this problem got solved by the proposed algorithm, we consider the rear door depicted in figure ***(Fig. 3.13)*** as an example. The red bars within the image mark the measurement spots and denote the orientation of the laser line.

[16] The weight vector is part of the position calculation and will be covered within the next section.
[17] "fully qualified" means a 6 dimensional vector combined of 3 components for the position and another 3 for the orientation

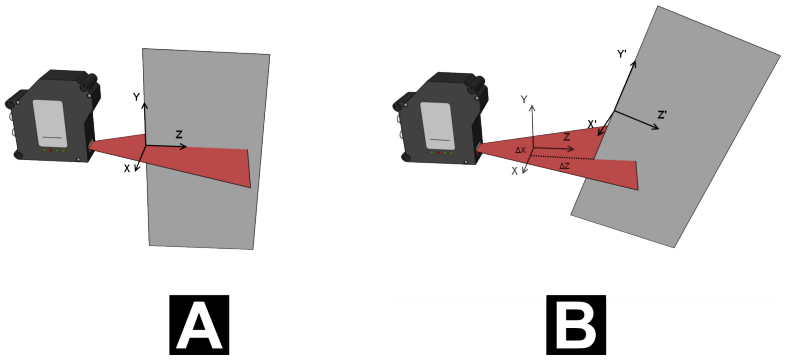

Figure 3.12: Sub image A depicts the original position during set up. Sub image B depicts the scene after the part got shifted the sensor moved slightly.

If the door now moves, all measures might change in a similar way as illustrated in figure ***(Fig. 3.12)***. Image ***(Fig. 3.14)*** shows all relations at a single measurement spot when the part has moved. One will be able to observe a change of $(\Delta x, \Delta z)$, but the information actually desirable is the transformation $T_{S_n}^{S'_n}$, which is giving the actual movement of the measurement spot. Unfortunately, this information is still unknown. It could, however, be expressed if one would know how far away from the original position of measurement the new measurement within the sensor frame $P' = (\Delta x, 0, \Delta z)^T$ is. This quantity is shown by $\Delta y'$ within figure ***(Fig. 3.14)***. By employing $\Delta y'$ the following relation of equation ***(3.44)*** could be assembled.

$$T_{S'_n}^{S_n} \cdot P' = \begin{pmatrix} 0 \\ \Delta y \\ 0 \end{pmatrix} \tag{3.44}$$

In figure ***(Fig. 3.15)***, both doors are visible, the shifted one in dark and the original position drawn brighter. The resulting shifted measurement spots, denoted as S'_n, are also illustrated in figure ***(Fig. 3.15)***. The transformations of the measurement spots T_{Sn}^{WO} are well-known, since they are the robot poses for the reference measurements S_n ***(Fig. 3.13)***. For the first measurement pose S_1 the position where the edge of the shifted door will be detected is marked as point P'. For simplicity, we model the local region around the reference position to be *linear* and therefore we can assume this measurement point P'_1 to be directly on the y-axis of the still unknown, shifted coordinate frame S'_1. The system of homogeneous transformations in between the "work object" coordinate systems and the sensor positions is illustrated in figure ***(Fig. 3.15)*** as well.

The desired transformation is the transformation between the referenced work object and the shifted work object $T_{WO'}^{WO}$. And, since we assumed that the sensor's y-axis is aligned to the contour shape of the object, as well as that P' is a measurement of the shifted contour, we can expect P' to be at the position Δy with $z =$ and $x = 0$ in the shifted coordinate system $S1'$. So we can expect the relation between $T_{S1'}^{S1}$ and P' to be as stated in equation ***(3.44)***. Unfortunately, the desired Δy is not observable since the edge of the door provides

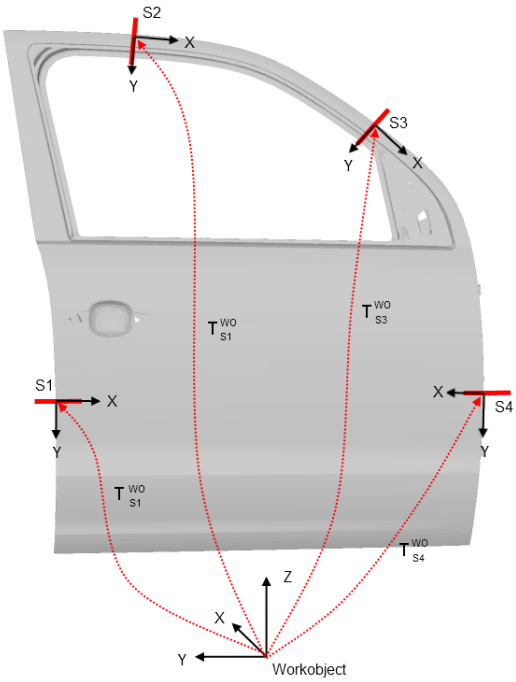

Figure 3.13: A door with possible measurement positions and according transfomations.

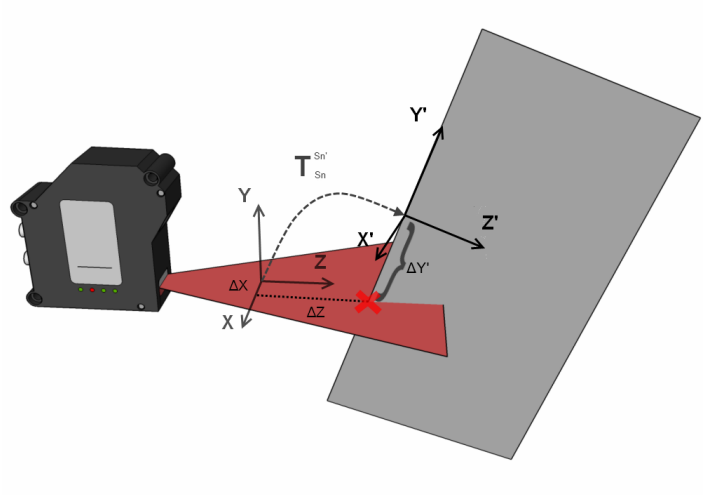

Figure 3.14: All relations at the measurement position with a shifted part.

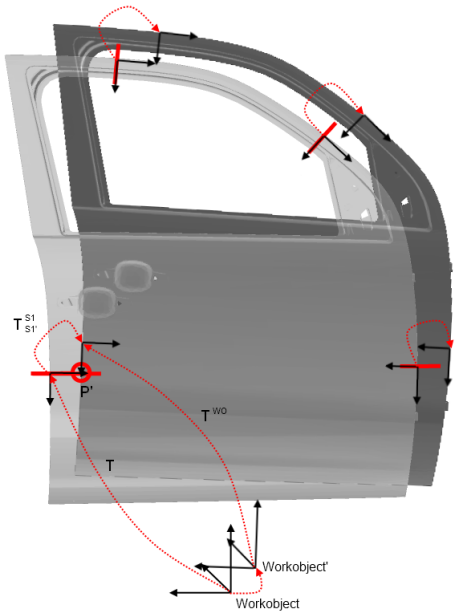

Figure 3.15: Exemplary shift with transformation for sensor 1.

no informations about this direction, nor is the sensor able to measure in y-direction. In other words, since the measurement at this position does not provide any information on the y-direction, this missing y quantity should have no influence on the calculation results. To achieve this, a weight matrix as given in equation ***(3.45)*** will be introduced.

$$\begin{aligned} & diag\left[\begin{pmatrix}\underline{w}\\1\end{pmatrix}\right]\cdot T_{S1'}^{S1}\cdot P' = \\ & diag\left[\begin{pmatrix}\underline{w}\\1\end{pmatrix}\right]\cdot\begin{pmatrix}0\\\Delta y\\0\end{pmatrix} \\ & with \qquad \underline{w}=\begin{pmatrix}w_x\\w_y\\w_z\end{pmatrix} \end{aligned} \tag{3.45}$$

Now, one can simply set the weight of the unobservable degree of freedom - which in this example is the y degree of freedom, but which could be any other degree of freedom depending on the scenario, to zero. This leads to equation ***(3.46)***.

$$diag\left[\begin{pmatrix}1\\0\\1\\1\end{pmatrix}\right]\cdot T_{S1'}^{S1}\cdot P' \quad =\begin{pmatrix}0\\0\\0\\1\end{pmatrix} \tag{3.46}$$

An expression for the transformation $T_{S1'}^{S1}$ can be extracted from figure ***(Fig. 3.15)***. As the transformation from the work object to the measurement spot is rigid, it follows that $T_{S1'}^{WO'} = T_{S1}^{WO}$. This assumption leads to the general description of the measurement position n as given in ***(3.47)***.

$$T_{S_n'}^{S_n} = \left(T_{S_n}^{WO}\right)^{-1} \cdot T_{corr} \cdot T_{S_n}^{WO} \tag{3.47}$$

Since all relationships have been described, a minimizer for an arbitrary set of measurements can be constructed. Equation ***(3.48)*** illustrates such a minimizer for N measurement spots, with Y as the unobservable degree of freedom and X and Z fully weighted in each position.

$$\sum_{n=1}^{N} \left\| diag[1\,0\,1\,1] \cdot \left[\left(T_{S_n}^{WO}\right)^{-1} \cdot T_{corr}(\theta) \cdot T_{S_n}^{WO} \right] \cdot \underline{P}_n' \right\|_2^2 \rightarrow min \tag{3.48}$$

Since the correction vector T_{corr} is the element to be adjusted by the solver, we added the parameter vector θ. The vector θ itself consists of the 6 adjustable degrees of freedom $\theta = (x, y, z, \alpha, \beta, \gamma)^T$. Up to this point we assumed that the reference is at the origin of the sensor coordinate system. To be more general, we now expect the reference point in the reference measurement to be at P_{ref}. Therefore the general error functions with independent weight vectors $\underline{w}_n$ for each measurement position, where $\underline{w}_n \in W$ is the set of all weights, is given by equation ***(3.49)***.

$$f(\theta\,|W\,) = \sum_{n=1}^{N} \left\| diag\left[\begin{pmatrix}\underline{w}\\ 1\end{pmatrix}\right] \cdot \left(\underline{P}_{ref,n} - \left[\left(T_{S_n}^{WO}\right)^{-1} \cdot T_{corr}(\theta) \cdot T_{S_n}^{WO}\right] \cdot \underline{P}_n'\right) \right\|_2^2 \tag{3.49}$$

With function $f(\theta\,|W\,)$ ***(3.49)*** we now have an expression for the error between the actual measurement and a guessed position θ with a flexible weighting method. If a degree of freedom is unobservable with the given choice of sensor, the unobservable degree of freedom should be suspended by its weight. This is what has been done in ***(3.48)*** for Y.
With equation ***(3.49)***, one now has a nonlinear function describing the sum of squared deviations over all measurement poses for a given θ. In order to find a θ which minimizes the result $f(\theta\,|W\,)$, one needs to employ a nonlinear least square solver. For the related implementation of this algorithm *Levenberg Marquard* has been chosen, due to its positive stability properties. This solver is described in detail within appendix *[B]*.

Chapter 4

Applying the results for visual servoing

In visual servoing sensor live data will be used within a control loop as feedback information of the robot position. That means the robot will be guided iteratively to the target position based on the measures captured during the control process. Many visual servoing tasks, and especially those for placing wing parts, rely on laser stripe sensors. A drawback of such visual servoing systems, however, is their iterative nature. The implementation of such a system usually requires rather complex interfaces in order to command the robot step by step into the target position. And not only complexity is an issue, since it is clearly more time consuming to reach the target position within multiple small steps than in one single step.

Therefore, the subsequent chapter describes how the previously proposed method of path correction *[3.3]* can be adjusted to perform the positioning of wing parts in a single step.
For visual servoing, the robot usually holds the wing parts within its grabber. Hence, those grabbers are significantly bigger and much bulkier than the common tools for sealing applications. Further, it requires multiple sensors, which are fixed in the predetermined measurement positions. Figure ***(Fig. 4.1)*** depicts such a grabber. The sensors within this picture are arranged in a double-head formation, whose purpose is to offer much more detail about the inner structure of the contour than a single sensor would be able to. Each sensor is oriented to a single edge side and will be able to look deep inside the gap. Then, both images will be combined to a single image before further processing.

Although this arrangement might be beneficial for the placing process, one might easily imagine how hard it is to calibrate such bulky grabbers with this complex sensor arrangements. Usually, in common robot cells the space is simply not sufficient for methods as described in section *[2.4.3]*, since these methods need the sensor to move around a calibration target with sufficient angular changes. Therefore section *[4.1]* describes an alternative TCP-calibration method that is applicable without moving the robot.
Finally section *[4.2]* proposes an modification of the original path correction algorithm, introduced in section *[3.3]*, to perform the positioning of wing parts.

4.1 Calibration of the sensor in a single step

As described in the introduction, there is a demand for novel calibration methods, due to the dimensions and the complexity of the grabbers used for panel fitting. State-of-the-art visual servoing systems do not require accurate extrinsic calibration, since such systems usually rely on a simple stepwise approach based on Jacobian matrices [63]. However, the novel method proposed in section *[4.2]* is designed to place the panel within one single step. Hence a good knowledge about the sensor positions is crucial.

Similar to the calibration described in section *[2.4.3]*, the proposed method will determine the relation between the coordinate system within the robots wrist and the coordinate system related to the sensors origin. The difference, however, is that this needs to happen without moving the sensors, due to the previously described limitations within motion.
Hence, it will be possible to calibrate a grabber, like the one illustrated in figure ***(Fig. 4.1)***,

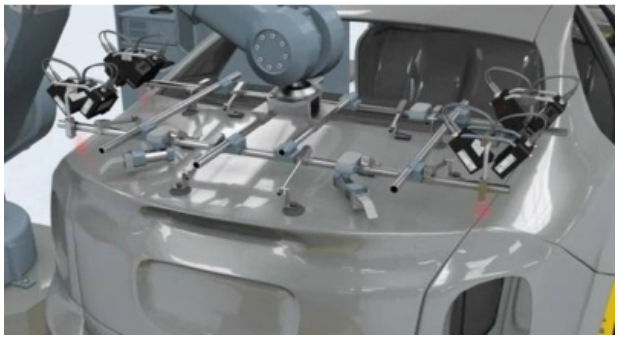

Figure 4.1: Positioning of a trunk lid

which is a typical grabber for fitting a trunk lid, in one single position in front of a calibration target.
To be able to perform a single-shot calibration, the proposed algorithm utilizes the internal camera of the sensor[1]. For most of the common laser line sensors, next to the extracted laser line coordinates, it is also possible to acquire the raw camera images. Since a camera can easily be calibrated by taking a single shot of a calibration plate, one can determine the camera's coordinate system by acquiring a single image of a well-known calibration plate in world coordinates. By having the camera's coordinate systems origin, the last thing one needs to determine is the relation between the sensor's coordinate system and the camera's coordinate system. By having the calibrated pinhole camera model, one can calculate the laser planes origin by positioning a well-defined calibration obstacle underneath the laser in

[1] Refer to section *[2.4]* for details of the sensors working principle.

various positions. Then, by knowing the relation between the camera and the laser planes origin, as well as the camera's position over the calibration plate, one will be able to calculate the relation from the robot's wrist to the sensor coordinate system. Image ***(Fig. 4.2)*** illustrates the interaction between the coordinate systems. The determination of the camera-

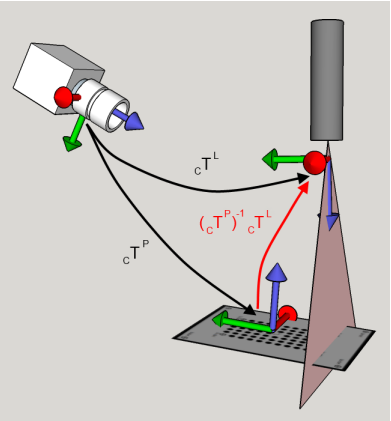

Figure 4.2: relation between the calibration coordinate systems

laser relation needs to be done before the sensor is mounted in its final position. This step still includes motion and thus a calibration robot is used before the sensor is supplied to the customer. After that, when the sensor is mounted, there will be a calibration rack containing all the calibration plates such that the robot can find a position where all sensors can observe their related plate. Now by making a single camera image of the plates and by having the previously determined camera laser relationship the sensors can be calibrated within a single shot.

4.1.1 Calibrating the lens distortion

To calibrate the distortion of the lens, we used the model proposed by Horn [64]. This lens distortion model is principally based on *Brown's distortion model* [7], which was introduced in section *[2.1.4]*. As suggested in Horn [64], we used the second order radial distortion formula, since that should be sufficient for optical systems. Furthermore, we are not including the tangential distortion, since that is only important within electro-optical systems [64]. Thus, the distortion model is given by equations ***(4.1)*** and ***(4.2)***.

$$u_d = u_u(1 + K_1 r^2 + K_2 r^4) \quad (4.1)$$

$$v_d = v_u(1 + K_1 r^2 + K_2 r^4) \quad (4.2)$$

$$with \quad r = \sqrt{(u_u - u_0)^2 + (v_u - v_0)^2}$$

Within equation ***(4.1)*** and ***(4.2)*** u_d and v_d are the distorted-image coordinates, and therefore u_u and v_u need to be the undistorted counterparts. The two parameters of the distortion function K_1 and K_2 are the two unknown parameters, which we want to solve for. Since

both equation ***(4.1)*** and ***(4.2)*** pose a non-linear problem, we need to approximate those parameters with an iterative algorithm. Such an iterative algorithm usually needs a cost function which provides some kind of feedback about the quality of the actual state within the iteration. A common approach is to use linear shapes within the scene to calibrate the distortion. For laser triangulation sensors one can achieve this by just placing a planar object underneath the laser, which will result into a line within the sensor image. Therefore, holding such a planar object in various positions and orientation underneath the laser, will create an image of multiple lines. Due to the lens distortion, these lines will appear with a slight curvature within the image. The distortion calibration parameters are then chosen in a way that ensures that all lines appear as straight as possible.

To solve for the unknown parameters within non-linear equations ***(4.1)*** and ***(4.2)***, we used a *Levenber-Marquardt* solver [65]. And to provide a cost-function for the solver, which measures the linearity of the lines within the image, we applied a *Karhunen-Loeve transformation* [66]. Therefore, we build the covariance matrix of the two-dimensional set of a single line's measures and solve for its Eigenvalues. The larger of the two resulting Eigenvalues λ_{max} is the one which is pointing into line direction, whereas the smaller λ_{min} is orthogonal to the line direction. The more the points are correlated in a linear way, the bigger λ_{max} gets, whereas on the other hand λ_{min} is getting smaller. By observing the quotient of $\lambda_{min}/\lambda_{min}$ we expect the best parameter for $K_1 \quad \& \quad K_2$ to be at the point where the quotient of $\lambda_{min}/\lambda_{min}$ gets the smallest. Let's consider L_i to be a $2 \times n$-Matrix containing the n measures of the i^{th} line. Then both values λ_{min} and λ_{min} are calculated as illustrated in equations ***(4.3)*** & ***(4.4)***.

$$\begin{aligned} L_i &= \begin{pmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \end{pmatrix} \\ &\Rightarrow \quad \lambda_{min} = min(eig(L_i L_i^T)) \qquad (4.3) \\ &\Rightarrow \quad \lambda_{max} = max(eig(L_i L_i^T)) \qquad (4.4) \end{aligned}$$

4.1.2 Camera model

Since all measures of a laser triangulation sensor are within the laser plane, usually no complex camera models are consulted to calibrate the relationship. Thus a calibration for a laser triangulation sensor is usually based on a simple two-dimensional homography matrix, often denoted as H. Let's consider the two point sets X and $\hat{X}$, where $\hat{x}_i \in \hat{X}$ are the projections of some observed features within the camera frame and $x_i \in X$ are the measures of the same features within the laser frame. One will be able to transfer each point from the camera plane to the laser plane by simply multiplying with the correct H-matrix, in a way that all vectors are parallel but of different scale. However, since the goal is to determine the extrinsic parameters of the sensor, one needs a representation where all parameters are separated and not just a homography matrix H. Therefore, equation ***(4.5)***, which is based on Tasi camera calibration [67], was chosen to model the point relations.

$$\begin{pmatrix} \hat{u}_i \\ \hat{v}_i \\ \hat{w}_i \end{pmatrix} = C \cdot (I_{3\times3}|0_{3\times1}) \cdot \left(\begin{array}{c|c} \mathbf{R} & \mathbf{t} \\ \hline \mathbf{0} & 1 \end{array}\right) \cdot \begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} \qquad (4.5)$$

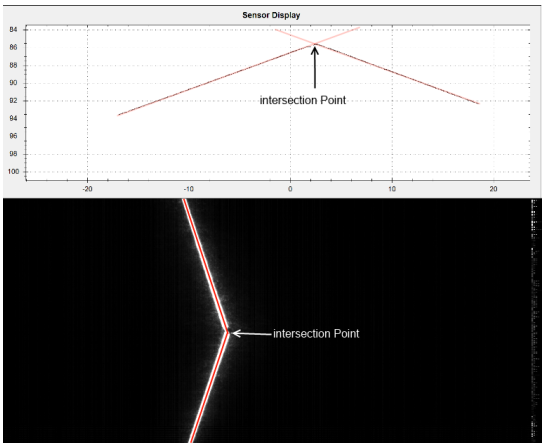

Figure 4.3: Top: Calibrated distance measures. Bottom: Raw camera image.

The resulting vector in equation ***(4.5)*** has a third component $\hat{w}_i$, which is the scale. Thus, to achieve the final pixel coordinates one needs to multiply the unscaled point $\hat{u}_i$ and $\hat{v}_i$ with its related scale factor $\hat{w}_i$.
The 3×3 Matrix C is the camera matrix, which is including all intrinsic parameters. It is composed as stated in equation ***(4.6)***.

$$C = \begin{pmatrix} f \cdot s_x & 0 & u_0 \\ 0 & f \cdot s_y & v_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.6}$$

The intrinsic parameters of C are well known at this point. The focus f is given by the lens, s_x and s_y are the pixel dimensions and can be found within the cameras data sheet and finally the center point coordinates u_0 and v_0 had been determined by the distortion calibration in equation ***(4.1)*** and ***(4.2)***.
The rotation matrix R is a 3×3 Matrix, which is composed by 3 angles, α, β and γ. The notation used within this paper is the *Tait-Bryan (YP'R")* notation and its rotation matrix is computed as in equation ***(4.7)***[2].

$$R = \begin{pmatrix} c(\alpha)c(\beta) & c(\alpha)s(\beta)s(\gamma) - s(\alpha)c(\gamma) & c(\alpha)s(\beta)c(\gamma) + s(\alpha)s(\gamma) \\ s(\alpha)c(\beta) & s(\alpha)s(\beta)s(\gamma) + c(\alpha)c(\gamma) & s(\alpha)s(\beta)c(\gamma) - c(\alpha)s(\gamma) \\ -s(\beta) & c(\beta)s(\gamma) & c(\beta)c(\gamma) \end{pmatrix} \tag{4.7}$$

And finally t is a 3×1 vector which holds the translation from the world origin to the camera coordinate systems origin in x, y and z. Since R and T can be composed by 3 parameters each, one can replace the matrix holding the extrinsic parameter of equation ***(4.5)*** by a function depending on these 6 parameters. And since all other parameters are well-known, we can replace the whole projection by a function holding these 6 parameters which is illustrated in

[2] The two function $c(x)$ and $s(x)$ within equation ***(4.7)*** are representing the cosine *(c)* and sine *(s)* functions.

equation ***(4.8)***.

$$\begin{pmatrix} \hat{u}_i \\ \hat{v}_i \\ \hat{w}_i \end{pmatrix} = P(x, y, z, \alpha, \beta, \gamma) \cdot \begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} \tag{4.8}$$

Now, having equation ***(4.8)*** one can calculate for any point in world coordinates $(x_i, y_i, z_i, 1)^T$ its related projection within the image plane. This can be achieved by multiplying $\hat{u}_i$ and $\hat{v}_i$ with the scale factor $\hat{w}_i$. Therefore the desired parameter vector $(x, y, z, \alpha, \beta, \gamma)$, is the vector which minimizes the function given in equation ***(A.13)***.

$$\underset{x,y,z,\alpha,\beta,\gamma}{\operatorname{argmin}} \left\| \begin{pmatrix} u_i \\ v_i \end{pmatrix} - \hat{w}_i \cdot \begin{pmatrix} \hat{u}_i \\ \hat{v}_i \end{pmatrix} \right\|_2^2 \tag{4.9}$$

4.1.3 Performing the calibration

The first step to determine the relation between the cameras and the laser planes origin is to determine the lens distortion. This can be achieved by following the description of section *[4.1.1]*. After the distortion has been estimated, one needs to determine the relation between the camera's coordinate system and the lasers' coordinate system. Therefore, we utilized a calibration body with triangular shape ***(Fig. 4.3)***. The calibration body has been placed in 10 positions within the sensor's field of view. The positions have been chosen in a way that the whole field of view was covered consistently. For each position, the sensor image and the cameras image have been recorded. After that, the two outer edges of the triangular body within the image pairs were extracted and approximated by lines. The intersection of these two lines, which is the bodies top edge, had been determined for each body position (***(Fig. 4.3)***.

The i^{th} intersection point within the camera had been denoted as $(u_i, v_i)^T$ and it's pendant within the sensor measure as $(x_i, 0, z_i)^T$[3]. Finally, all 6 unknown parameters can be solved by a *Levenber-Marquardt* algorithm in combination with the given cost-function of equation ***(A.13)***. The resulting parameters describe the transformation from the camera to the laser plane ${}_C T^S$. Figure ***(Fig. 4.2)*** illustrates this relationship. These two calibration steps can be performed anywhere, yet the sensor does not need to be mounted in its final position.

Now, since the relation between the camera coordinate system and the laser coordinate frame is known, one only need to find the position of the camera within the world coordinates. Therefore, the sensor needs to be mounted in its final position. To calibrate the final position, we used an ordinary calibration plate ***(Fig. 4.4)***. This calibration plate consists of well-known markers. Some circular markers with a black cross inside and some bold black circles. The bold black circles are the markers used to determine the transformation between the camera and the coordinate system of the calibration plate. The approach to determine this relation is exactly the same as for the calibration of the laser coordinate system. The

[3] Y is fixed to zero, since the Y-plane usually is the laser plane.

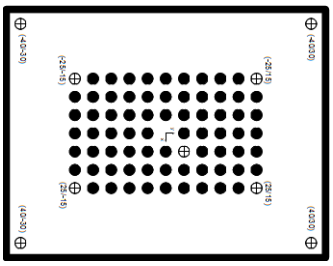

Figure 4.4: Plate used for calibration

coordinates $(x_i, y_i, z_i)^T$ of the bold circles within the calibration plate's coordinate system are well known and act as the reference points. For the scene point measures, the camera image of the plate has been taken and the centers of the bold black points $(u_i, v_i)^T$ have been extracted within this image. Having these two data sets, one can use the same function ***(A.13)*** to solve for the optimal transformation between the two frames.
This finally results into two transformations. One describes the relation between the camera and the laser ${}_C T^L$ and the other describes the relation between the camera and the calibration plate ${}_C T^P$. Therefore, one achieves the relation between the calibration plate and the laser ${}_P T^L$ by equation ***(4.10)***.

$$ {}_P T^L = ({}_C T^P)^{-1} \cdot_C T^L \tag{4.10} $$

This relation is also illustrated in figure ***(Fig. 4.2)***. Finally, the last thing one needs to determine is the position of the calibration plate within the world coordinate system.

4.1.4 Calibration of the plate position in world space

To determine the position of the calibration plate within the world coordinate system, the circles with the black cross on the calibration plate were used. Their position within the plate is well-known and thus has been used as reference.Having the cross position within the plate coordinate system, one also needs the related cross positions within the world coordinate system. This can be achieved by an external measurement system[4]. Now, having the two point-sets which are the reference or model points on the plate $p_i \in \mathbb{R}^3$ and the related measured points within the world coordinate system $p'_i \in \mathbb{R}^3$ one can determine the relation in-between the two point-sets by using the method described by K. Arun [68]. With this method, one can calculate the transformation in three steps. First, the translation vector between the two point clouds is the translation between the two centroids, thus one needs to calculate the centroids of the two point-sets.

$$ \bar{p} = \frac{1}{N} \sum_{i=0}^{N} p_i \quad , \quad \bar{p}' = \frac{1}{N} \sum_{i=0}^{N} p'_i \tag{4.11} $$

[4]Usually the robot where the sensor has been mounted was used for such an measurement

It follows then, that after subtracting the centroids from each point of the related point cloud, the two point sets differ only in orientation.

$$Q = \begin{pmatrix} p_1^T - \bar{p} \\ p_2^T - \bar{p} \\ \vdots \\ p_N^T - \bar{p} \end{pmatrix} \quad , \quad Q' = \begin{pmatrix} p'^T_1 - \bar{p}' \\ p'^T_2 - \bar{p}' \\ \vdots \\ p'^T_N - \bar{p}' \end{pmatrix} \tag{4.12}$$

$$\Rightarrow q_i = R \cdot q_i' \tag{4.13}$$

Thus, the last step is to determine the unknown ration matrix R, which can be calculated by equation ***(4.15)***.

$$[U, \Lambda, V] = svd\left((Q')^T Q\right) \tag{4.14}$$

$$R = V \cdot diag(1, 1, det(V \cdot U^T)) \cdot U^T \tag{4.15}$$

Finally, since the transformation from the scene[5] to the reference[6] has been determined, one needs to transform the center of mass of the scene points to the model's orientation and calculate the relative vector in between the two centers of masses.

$$t = \bar{p} - R \cdot \bar{p}' \tag{4.16}$$

$${}_W T^P = \left(\begin{array}{c|c} \mathbf{R} & \mathbf{t} \\ \hline \mathbf{0} & 1 \end{array}\right) \tag{4.17}$$

Equation ***(4.17)*** shows the relation between the scene and the model points, or, as in the case between the world-coordinate frame and the calibration plate's coordinate frame. Since we also derived the relation between the calibration plate's frame and the laser coordinate system in equation ***(4.10)***, the final step, which calculates the relation between the world and the lasers' coordinate system, is given by equation ***(4.18)***.

$${}_W T^L = {}_W T^P \cdot {}_P T^L \tag{4.18}$$

4.2 Fitting the panel in a single step

The panel-fitting algorithm proposed in this section is an extension of the path correction. The task is to fit a wing panel such as a fender, a door, the trunk lid or the motor hood relative to the car body. Due to the oversized grabber, one challenge is the sensor calibration, as discussed in section *[4.1]*. On the other hand, all panels will be placed within a frame, therefore the situation is quite similar, as illustrated in figure ***(Fig. 1.7)***. Since the frame and panel edges are usually very close to each other, they act as references and the slightest deviation will be easily spotted by the customer. For panels, this means they have to be corrected to a position where all gaps between the panel and the body look symmetrical to the observer.

[5] Which is the world coordinate system.
[6] Which is the plates coordinate system.

To do so, a relative measure is necessary, meaning a measure between body and panel, different from the path correction. Therefore, the algorithm proposed in this section introduces a new method to find a mounting position for panels within the body frame. It is based on the path correction introduced in section *[2.5]*. Other than for existing solutions, which are working iteratively, it is possible to find the final mounting position in a single step. To set up the system, the desired position for the panel fitting needs to be trained. This means the robot teacher needs to grab the panel and needs to position the robot to the final (zero) position.

This is the position where the panel needs to be mounted. This mounting position is not always the position where all of the gaps appear to be symmetrical, because after the grabber opens and the part is only held by its hinges, it will lower a bit into its final position. Hence this final position is not related to the mathematically perfect position, but to the outcome after fixation and relaxation of the grabber. As soon as the door has been positioned, the system starts to measure both the panels and the body frame.

To extract the absolute position for each coordinate system, the path correction algorithm (section *[2.5]*) will be utilized. Thus, the mounting points of the sensor need to be calibrated which we discussed in section *[4.1]*. After having the absolute measure of both coordinate systems, one can calculate the relative relation in between the panel and the frame. This relative relation is what we need to achieve for all misaligned doors. Hence, for a misaligned door, the correction vector is exactly the difference between this zero-relationship and the actual measured relationship.

4.2.1 An Offline Single-Step Algorithm

The algorithms proposed in [69] are designed to position car body panels in the so-called *"BestFit"*position. That is the position where all measurements have no deviations, or at least the smallest possible deviations with respect to the statistical model for the expected residuals. Since the statistical model for the expected residuals is usually a Gaussian, this is almost always the position where the summed squares of the residuals is minimal. Figure

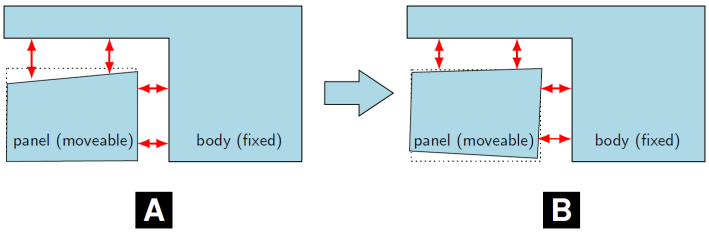

Figure 4.5: (A): Misplaced panel with deformations (B): *"BestFit"*position of deformed panel

(Fig. 4.5) sketches a scene where a deformed and misaligned panel *(subfigure A)* has been fitted into the *"BestFit"*position relative to the fixed body *(subfigure B)*. The final position in subfigure B relies on the four distance measures, which are indicated by red arrows. The

dotted-lined boxes in the background of the panels in both subimages indicate the original's panel shape, as well as the desired mounting position. To reach the mounting position for the given deformed panel, the closed-loop Jacobian approach would estimate improving positions step by step until the panel position converges.

The panel in sub-image B of ***(Fig. 4.5)*** shows the *"BestFit"*position, for which the panel has been moved from its starting position in sub-image A, through a well-defined correction transformation $\boldsymbol{T}_{corr}$. The correction transformation $\boldsymbol{T}_{corr}$ is the transformation where the difference between current sensor signals $\underline{s}\left(\boldsymbol{T}_{corr}\right)$ and the signals for the set-position $\underline{s}_{set}$ are at minimum ***(4.19)***.

$$\boldsymbol{T}_{corr} = \min_{\forall \boldsymbol{T} \in \mathfrak{T}} \left\| \underline{s}\left(\boldsymbol{T}\right) - \underline{s}_{set} \right\|_2^2 \tag{4.19}$$

$$\Delta \underline{s} = \underline{s}\left(\boldsymbol{T}_{corr}\right) - \underline{s}_{set} \tag{4.20}$$

Where $\mathfrak{T}$ is the set of all possible transformations in $\mathbb{R}^3$. Equation ***(4.20)*** expresses the deviations $\Delta \underline{s}$ in the final *"BestFit"*position, the so-called residuals.

If one considers a perfectly-shaped panel and a perfectly-shaped object with no deformations related to the object used during the training, those residuals would completely disappear in the final mounting position $\boldsymbol{T}_{corr}$. In other words, the residuals represent the deviations of the objects used for positioning from the objects which were used during training. Since two objects are involved in the control process – the body and the panel – the residuals in the final *"BestFit"*position $\Delta \underline{s}_{BestFit}$ are the sum of the deviations due to the body and the panel ***(4.21)***.

$$\Delta \underline{s}_{BestFit} = \Delta \underline{s}_{panel} + \Delta \underline{s}_{body} \tag{4.21}$$

$$\boldsymbol{T}_{corr} = \min_{\forall \boldsymbol{T} \in \mathfrak{T}} \left\| \Delta \underline{s}_{panel}\left(\boldsymbol{T}\right) + \Delta \underline{s}_{body}\left(\boldsymbol{T}\right) - \underline{s}_{set} \right\|_2^2 \tag{4.22}$$

The refined definition for residuals in ***(4.21)*** can be inserted in equation ***(4.19)*** to yield the expression in ***(4.22)***.

Section *[3]* presents a new method to determine the relative shift between a reference object and later measurements of similar but deformed and shifted objects, for use in path correction systems. That algorithm perfectly fits the demand to find the residuals at the panel $\Delta \underline{s}_{panel}$, as well as the body $\Delta \underline{s}_{body}$. The algorithm proposed in *[3]* requires no model information and is able to work on sparse dimensional measurements. That means it is not necessary to perform measurements for all degrees of freedom, since sparse 1D information – for example a gap distance in the case of a door – can be sufficient. Therefore, the position determination can be accomplished with basically the same signals considered in Roebrock [69]. The only restrictions are that the sensor's positions need to be calibrated and that the signals need to represent distances in real-world metric.

To introduce our new algorithm, we consider a *rear* door as an example. Such a door is illustrated in figure ***(Fig. 4.6)***. The red bars in the image mark the measurement spots and denote the orientation of the laser line. In a real-world situation one would use more

measurement spots than those given in figure ***(Fig. 4.6)***.

The proposed algorithm is expected to be able to find the relationship between a set of

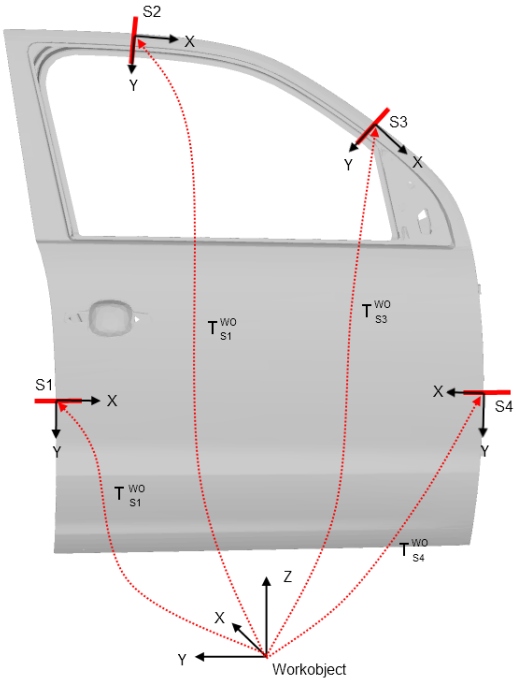

Figure 4.6: A door with possible measurement positions and according transfomations.

reference measurements and a set of new shifted measurements at arbitrarily shaped objects. A priori knowledge, for instance a CAD model of the object to be measured, is not required. After referencing the master position, we expect the door to be shifted and, since the measurement positions are fixed this implies that the measurements of a shifted position are then related to a slightly different spot at the door. This is illustrated in figure ***(Fig. 4.7)***, which shows a dark door and a bright one. The bright door is the original reference door, whereas the dark door indicates a shifted door position. The resulting shifted measurement spots, denoted as $\underline{s}'_n$, are also illustrated in figure ***(Fig. 4.7)***. The transformations of the measurement spots $\boldsymbol{T}^{WO}_{Sn}$ are well-known, since they are the robot poses for the reference measurements $\underline{s}_n$ ***(Fig. 4.6)***. For the first measurement pose $\underline{s}_1$ the position where the edge of the shifted door will be detected is marked as point $\underline{p}'$. For simplicity, we model the local region around the reference position to be *linear* and therefore we can assume this measurement point $\underline{p}'_1$ to be directly on the y-axis of the still unknown, shifted coordinate frame $\underline{s}'_1$. The system of homogeneous transformations in between the "work object" coordinate systems and the sensor positions is illustrated in figure ***(Fig. 4.7)***, as well.

The desired transformation is the transformation between the referenced work object and the shifted work object $\boldsymbol{T}^{WO}_{WO'}$. And, since we assumed that the sensor's y-axis is aligned to the contour shape of the object and that $\underline{p}'$ is a measurement of the shifted contour, we can

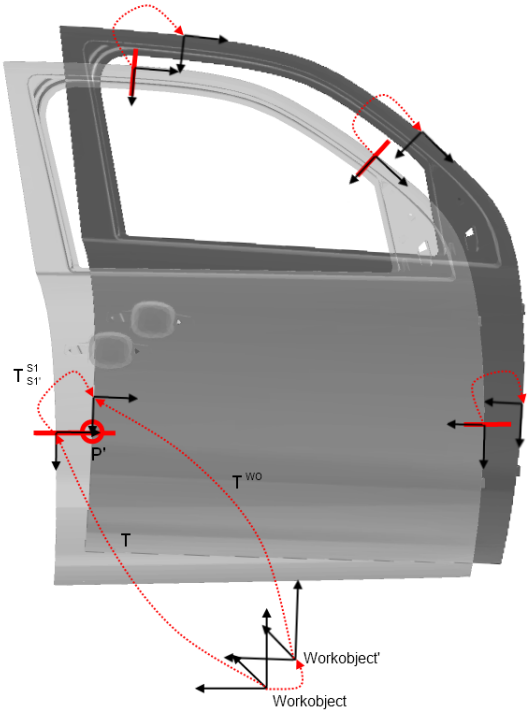

Figure 4.7: Exemplary shift with transformation for sensor 1.

expect $\underline{p}'$ to be at the position Δy with $z =$ and $x = 0$ in the shifted coordinate system $\underline{s}1'$. So we can expect the relation between $\boldsymbol{T}_{S1'}^{S1}$ and $\underline{p}'$ to be as stated in equation ***(4.23)***.

$$\boldsymbol{T}_{S1'}^{S1} \cdot \underline{p}' = \begin{pmatrix} 0 \\ \Delta y \\ 0 \end{pmatrix} \tag{4.23}$$

Unfortunately, the desired Δy is not observable since the edge of the door provides no information about this direction, nor is the sensor able to measure in y-direction. In other words, since the measurement at this position does not provide any information on the y-direction, this missing y-quantity should have no influence on the calculation results. To achieve this,

we will introduce a weight matrix as in equation ***(4.24)***.

$$\begin{aligned} &diag\left[\begin{pmatrix}\underline{w}\\1\end{pmatrix}\right]\cdot \boldsymbol{T}_{S1'}^{S1}\cdot{}' = \\ &diag\left[\begin{pmatrix}\underline{w}\\1\end{pmatrix}\right]\cdot\begin{pmatrix}0\\\Delta y\\0\end{pmatrix} \\ &with \qquad \underline{w}=\begin{pmatrix}w_x\\w_y\\w_z\end{pmatrix} \end{aligned} \tag{4.24}$$

Now one can simply set the weight of the unobservable degree of freedom to zero. In this example, this is the y-degree of freedom but it could be any other degree of freedom depending on the scenario. This leads to equation ***(4.25)***.

$$diag\left[\begin{pmatrix}1\\0\\1\\1\end{pmatrix}\right]\cdot \boldsymbol{T}_{S1'}^{S1}\cdot \underline{p}' \quad = \begin{pmatrix}0\\0\\0\\1\end{pmatrix} \tag{4.25}$$

An expression for the transformation $T_{S1'}^{S1}$ can be extracted from figure ***(Fig. 4.7)***. As the transformation from the work object to the measurement spot is rigid, it follows that $T_{S1'}^{WO'} = T_{S1}^{WO}$. This assumption leads to the general description of the measurement position n as given in ***(4.26)***.

$$\boldsymbol{T}_{S_n'}^{S_n} \quad = \quad \left(\boldsymbol{T}_{S_n}^{WO}\right)^{-1}\cdot \boldsymbol{T}_{corr}\cdot \boldsymbol{T}_{S_n}^{WO} \tag{4.26}$$

So far, we have described all necessary relationships to derive a function for the expected deviation. Equation ***(4.27)*** illustrates such a minimizer for the n-th measurement spot, with Y as the unobservable degree of freedom and X and Z fully weighted in each position.

$$f_{\Delta \underline{s}_n}(\boldsymbol{T}) = diag[1\,0\,1\,1]\cdot\left[\left(\boldsymbol{T}_{S_n}^{WO}\right)^{-1}\cdot \boldsymbol{T}\cdot \boldsymbol{T}_{S_n}^{WO}\right]\cdot \underline{p}_n' \tag{4.27}$$

First, the matrix $\boldsymbol{T}$ is rearranged into a function depending on the parameter vector $\underline{\theta}$. The vector θ itself consists of the six adjustable degrees of freedom $\underline{\theta} = (x, y, z, \alpha, \beta, \gamma)^T$. Up to this point, we assumed that the reference is at the origin of the sensor coordinate system. To generalize the approach, the reference point in the reference measurement is expected to be at $_{ref}$ instead. Therefore the general error functions with independent weight vectors $\underline{w}_n$ for each measurement position, where $\underline{w}_n \in W$ is the set of all weights, is given by equation ***(4.28)***.

$$\begin{aligned} f_{\Delta \underline{s}_n}(\underline{\theta}\,|\underline{w}_n) = diag\left[\begin{pmatrix}\underline{w}_n\\1\end{pmatrix}\right]\cdot\Big(\underline{p}_{ref,n} - \Big[\left(\boldsymbol{T}_{S_n}^{WO}\right)^{-1} \\ \cdot f_{\boldsymbol{T}}(\theta)\cdot \boldsymbol{T}_{S_n}^{WO}\Big]\cdot \underline{p}_n'\Big) \end{aligned} \tag{4.28}$$

With the function $f_{\Delta \underline{s}_n}(\underline{\theta}\,|\underline{w}_n)$ derived in ***(4.28)***, the residuals for a given parameter vector $\underline{\theta}$ can be calculated. Thus, the residuals in the minimizer for the *"BestFit"*position, as given in ***(4.22)***, can be replaced with the function $f_{\Delta \underline{s}_n}(\underline{\theta}\,|\underline{w}_n)$ from equation ***(4.28)***. This leads to a new equation for the function to minimize, as illustrated in ***(4.29)***.

$$f_{\Delta \underline{s}}(\underline{\theta}\,|\mathfrak{W}) = \sum_{n=1}^{N} \Big\| f_{\Delta \underline{s}_{panel_n}}(\underline{\theta}\,|\underline{w}_n) + f_{\Delta \underline{s}_{body_n}}(\underline{\theta}\,|\underline{w}_n) - \underline{s}_{set} \Big\|_2^2 \tag{4.29}$$

Where $\mathfrak{W}$ is the ordered set of N weight vectors $\underline{w}_n$. Equation ***(4.29)*** now states a function describing the residuals for a given pose $\underline{\theta}$. The $\underline{\theta}$, which minimizes ***(4.29)***, is the desired pose of the *"BestFit"*-position for the panel. This resulting pose is related to the measurement position. To find such a minimizing $\underline{\theta}$ for equation ***(4.29)***, one needs to employ a nonlinear least square solver. For the related implementation of this algorithm a *"Levenberg-Marquardt"* solver [70] was chosen, due to its positive stability properties.

Chapter 5

Discussion and evaluation of the model accuracy

Vision systems for robot guidance usually assist the production process by measuring parts, calculating corrections for those parts and supply the corrections to the robot. After this, the robot will be able to correct its trajectory according to the correction vector provided by the vision system, which will result in better product quality. But when one talks about measurements, he also needs to talk about uncertainty, since no measurement is error-free.

On the one hand, one has to differentiate between two types of uncertainty, which can be random or systematic. A systematic error is within the applied mathematical model. Sometimes a mathematical model would simply get too complex if it reproduces the reality perfectly. Therefore, it is preferable to introduce some shortcomings to keep the model simple and manageable. Modeling all the factors that influence the realistic behavior would simply be too complex. The influence of different aspects that have been left out is unmodeled systematic behavior and can therefore be estimated. An other aspect are the purely random errors, which are induced by multiple environmental influences as well a technical limitation while recording the desired quantities from the system. An introduction to the nature of errors distribution and how to estimate optimal model parameter can be found in the appendix *[A.1]*

Contrary to systematical errors, a random error is not describable at all for a single value; only the way it is distributed like the density function or other stochastic features can be estimated. If one considers the camera model, as discussed in chapter *[2.6]*, then one has a model which is very accurate and where the systematic error, which might be within the thin lens and distortion model, is negligible for multiple types of measurements. As a simple rule of the thumb:

> The better the cameras resolution and noise ratio and the less the lense distorts (which can also be achieved by modeling see *[2.1.4]*), the more accurate the results[1]

[1] Of course this is only true for a certain range, but with right lense calibration and a good camera one

Since the proposed algorithms describe only the model functions but can be applied to multiple devices with different accuracies, the rule as described above holds in the same way. This means that for a four-megapixel camera, the accuracy almost doubles compared to a one-megapixel camera. But there also is a systematic error within the path-correction model, which will be discussed in section *[5.1]*. Within this section, the error will be approximated by a theoretical worst-case function.

5.1 Model inaccuracy

The weakness of the algorithm detailed in chapter *[3]* is that it is based on the assumption that the observed feature continues linearly in all unobservable degrees of freedom. This condition is rarely met in modern car bodies, since they have a lot of non-straight contour lines and curvatures. Therefore, one has to expect errors in the results of the position determination because the conditions of the real world do not match the theoretical model. Thus, an elementary question is how severely those deviations influence the results.

Since the accuracy of an ordinary industrial robot can be expected to be about $\pm 0.3mm$, the goal of this section is to determine the critical limit for the surface curvature that would not influence the resulting accuracy of the application. In this section, we will carve out a correlation between the shapes' curvature and the resulting error of the model assumption.

Since the mounting positions of panels usually do not allow a lot of movement without provoking a collision, the misplacement of panels is expected to not exceed $\pm 10mm$. That assumption is realistic, because it is a reasonable field of view for the sensor devices as well. One can expect the curvature to *not* change drastically within the allowed region of $\pm 10mm$. Therefore, we chose an osculating circle at the referenced measurement point as a sufficient approximation of the local curvature. Figure ***(Fig. 5.1)*** shows a car body door with a laser line sensor's measuring position at the front edge of the door and the related osculating circle for the given position.

The radius of the osculating circle is marked as r_{oc} in the image, $\underline{p}$ marks the referenced measurement position. The question is how big the maximum relative approximation error δ_p between the osculating circle and the linear model is within a given region ϵ_p. A sketch of this configuration is shown in ***(Fig. 5.3)***. For the scene as depicted in ***(Fig. 5.3)***, δ_p turns

can achieve accuracy within the range of a hundredth of a millimeter.

out to be as given in ***(5.1)***.

$$\delta_p = r_{oc} - \sqrt{r_{oc}^2 - \left(\frac{\epsilon_p}{2}\right)^2} \quad \Leftrightarrow \tag{5.1}$$

$$(r_{oc} - \delta_p)^2 = r_{oc}^2 - \left(\frac{\epsilon_p}{2}\right)^2 \quad \Leftrightarrow \tag{5.2}$$

$$r_{oc}^2 - 2r_{oc}\delta_p + \delta_p^2 = r_{oc}^2 - \left(\frac{\epsilon_p}{2}\right)^2 \quad \Leftrightarrow \tag{5.3}$$

$$2r_{oc}\delta_p - \delta_p^2 = \left(\frac{\epsilon_p}{2}\right)^2 \quad \Leftrightarrow \tag{5.4}$$

$$r_{oc} = \frac{\left(\frac{\epsilon_p}{2}\right)^2 + \delta_p^2}{2\delta_p} \tag{5.5}$$

With equation ***(5.5)***, equation ***(5.1)*** can be rearranged in such a way that the minimum osculating circle radius r_{oc} for a given relative approximation error δ_p can be determined. Figure ***(Fig. 5.2)*** also shows a sketch of the relation between the relative approximation error δ_p and the osculating circle radius r_{oc}. In equation ***(5.5)*** and figure ***(Fig. 5.2)***, it is

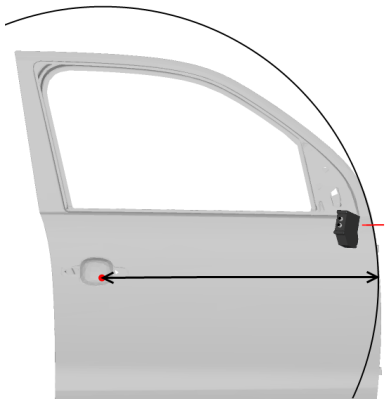

Figure 5.1: Measurement positions at a door with the corresponding osculating circle.

evident that the relative approximation error δ_p is quickly decreasing by a rising osculating circle radius r_{oc}. If one considers an osculating circle radius r_{oc} of about 1 meter for instance (which is almost always true for regular measurement spots) a maximum approximation error of about $0.05mm$ within a range of $20mm$ is achieved. This falls below the measurement accuracy of most sensor devices and also below what a common robot can compensate within its accuracy. Even if one would consider extreme shapes like a wheelhouse, an approximation error of about $\delta_p = 0.14mm$ for a given osculating circle radius $r_{oc} = 350mm$ within a range of $20mm$ can be achieved, which usually is an acceptable deviation.

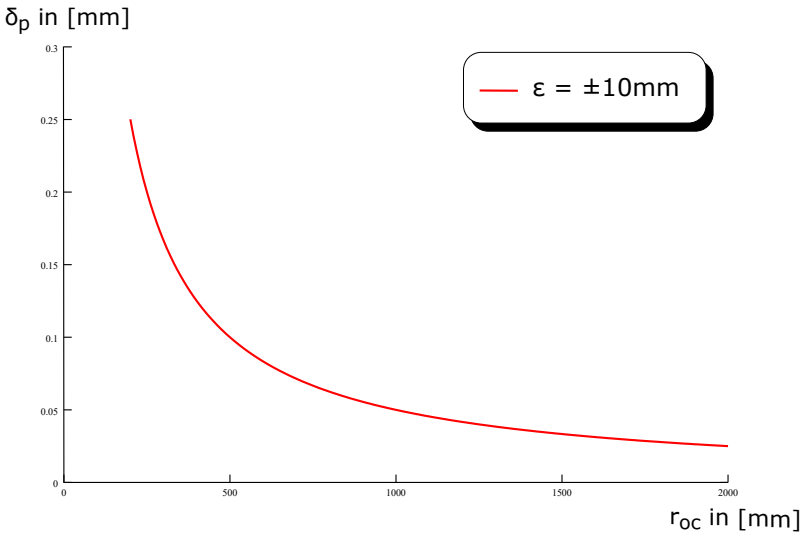

Figure 5.2: Shows the relation between the maximum possible error and osculating circle radius.

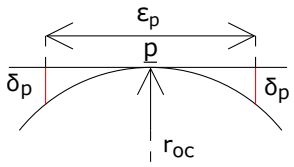

Figure 5.3: Sketch of relative approximation error around reference position.

5.2 Analysis and Tests

Despite the mathematical analysis it is important to observe and evaluate the practical performance of the proposed techniques. Since this thesis was developed in context of an industrial project at "VMT Bildverarbeitungssysteme GmbH" for integration of their path correction application "VMT BK" and is in production at major automotive companies since a few years, there is a lot of practical experience and the a major part of the proposed techniques approved reliability and functionality in practice.

The "VMT BK" system works in multiple production lines at Audi, Volkswagen, Porsche, Mercedes and Audi, to name just a few. It is also in use at multiple automotive supplier and some other industries. According to the VMT brochures and homepage the system is promoted with an in production accuracy of $0.1mm$. Although this accuracy had been achieved multiple times, it is important to monitor some key factors to achieve this quality, otherwise the accuracy might also be far below $0.1mm$. A few of these factors have already been mentioned earlier, like the shape of the edge, the alignment of the sensor, the locally parallel

trajectory of the robot path and finally the accuracy of the utilized sensor[2].

To validate the quality of the calculated corrections a first test setup had been created. In this test setup the measurements have been performed on a raw production body around the door frame. The shift of the object had been simulated by a base shift of the robot and the resulting measurements have been recorded for multiple shift positions, in which each of the measures had been executed at different positions. These positions were a set of linear translation from -10mm to 10mm in all three degree of freedom, after that rotation from -1°to 1°and finally both combined.

Finally, in order to get an idea about the exact accuracy and how much it gets affected from a poor setup, the system had been reviewed under "laboratory" conditions. The test setup was a single door on a moveable stand in front of a Kuka KR30. The door itself was equipped with measurement fixtures which can hold a 1,5" CCR reflector ***(Fig. 5.4)***. The position

Figure 5.4: 1,5" CCR Red Ring Reflector

of such reflectors can be determined very high accuracy by a device called "laser tracker". Depending on the type of "laser tracker" and the distance to the object[3] the measurement accuracy can be expected about $0.02mm$, which is quite sufficient as a reference measure. With this kind of setup the goal was to perform the following measurement experiments:

1. Standard setup:
   Like in a normal production, the positions T_{Sn}^{WO}[4] will be provided by the robot. Sensors are well aligned and trajectory runs locally parallel on the edge at trigger time. Measurement results will be compare with measured references from the "laser tracker"

2. Standard setup (not continuous):
   Equal to "Standard setup" but the sensor images will be recorded while the robot stands still at each measurement spot. In "Standard setup" the robot is moving continuously and just triggering at the spots.

3. Standard with improved feature detection:
   This setup is equal to the first setup, but with the improvements proposed in chapter

[2] No one can expect the system to measure far below $0.1mm$ if the utilized sensor is only capable the achieve $0.1mm$ accuracy.
[3] Which is quite close under such laboratory conditions
[4] From ***(4.28)*** in chapter *[3.3]* the Sensor position in the workobject.

[3.2] and *[3.1]* activated.

4. Misaligned sensor positions:
   This setup is intended to figure out how much the sensor alignment matters. It the necessary "local" parallelism between the sensor and the edge will be canceled, to see how much poor setup will influence the result quality.

5. Misaligned sensor positions (not continuous):
   Same like "Misaligned sensor positions" but with no continuous motion at the measurement spots.

5.2.1 Standard setup

To have an initial benchmark, the system was be configured in the standard setting. That means "local" parallelism at the measurement spot, generous chosen entry and exit positions, carefully monitored "frozen"[5] contour images at dynamic trigger positions, and stable edge features at straight contour lines. The "set-values" of the shift positions are determined by the laser tracker. Image ***(Fig. 5.5)*** depicts the absolute deviations of the three test sets. Sub image 1 displays the magnitude of the average deviation at each measurement spot after correction. The start position of the measurement (misplacement) is figured out on the axis of abscissas, going form -10mm to 10mm with steps of half a millimeter. The shifts themselves had been realized by base shifts, as mentioned above. This is the only way to grant somehow accurate starting positions, since the door and the frame can hardly be moved. Sub image 2 presents the same but for rotation from -1°to 1°. And finally the last image, which is sub image 3, which is showing the absolute deviations after corretion from translation combined with a rotation. All graphs are showing pretty stable results, within the range of $\pm 0.35mm$. In addition for this first test all deviations at the measurement spots were evaluated at the position at 5mm translation and 0.5°rotation displacement. The values are depicted in figure ***(Fig. 5.6)*** as histograms with the according estimations of the related Gaussian distributions overlayed. All samples represent the deviations between the forward transformation of the detected feature position within the sensor frame forward transformed by the determined correction after measurement and the "set" positions on the door frames. Hence the deviations were determined as 3 dimensional in (ΔX,ΔY,ΔZ). The results of the first test setup can be found in table 5.1. The mean values are almost at zero, as they should

Table 5.1: Test results of test 1

	Mean	Min	Max	Var	3σ
Sub Fig. 1	0,0573 mm	-0,1570 mm	0,6748 mm	0,0120 mm	0,3284 mm
Sub Fig. 2	0,0517 mm	-0,2049 mm	0,6504 mm	0,0150 mm	0,3672 mm
Sub Fig. 3	0,0572 mm	-0,1570 mm	0,6748 mm	0,0119 mm	0,3274 mm

be. But the distribution of the min and max values seems to be a little biased, since it tends

[5]A procedure during setup of the vision system is to slowly run the robot from the entry point over the trigger point to the exit point, while monitoring the display of the contour. If the system is well configured, this contour is not moving at all, while the robot tuns through the local parallel spot. Hence it's called a frozen contour.

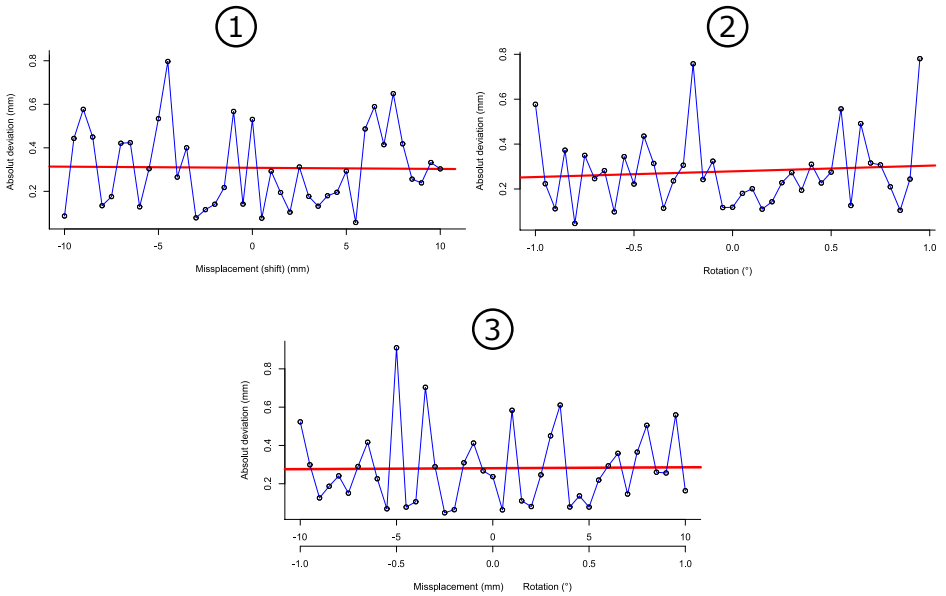

Figure 5.5: Test results for Test 1 with translation only in sub image 1, rotation only in sub image 2 and translation with rotation in sub image 3

to drift more in positive direction as in negative. But this effect might mostly be related to the relatively small sample size, and can hence be expected as normal distributed. And finally the resulting 3σ accuracy settles stable underneath half a millimeter, which is a really good result.

5.2.2 Standard setup (not continuous)

The second test setup is almost similar to the first, with the tiny difference that the robot is not moving when the measurements are taken. Finally due to the nature of the calculation, this should not make much of a difference. As described chapter *[3.3]*, there is a zero weight for the vector $\underline{w}$ in the motion direction, as it is used in formula ***(4.28)***. Hence if the system is configured carefully, the moving robot in the first test should not have a negative influence and one could expect similar results for this second test setup. Table 5.2 shows the results

Table 5.2: Test results of test 2

	Mean	Min	Max	Var	3σ
Sub Fig. 1	0,0514 mm	-0,1606 mm	0,6683 mm	0,0192 mm	0,4157 mm
Sub Fig. 2	0,0306 mm	-0,4265 mm	0,6879 mm	0,0194 mm	0,4177 mm
Sub Fig. 3	0,0482 mm	-0,1732 mm	0,6388 mm	0,0150 mm	0,3670 mm

of the second test and figure ***(Fig. 5.7)*** the graphs with the magnitudes of the resulting deviations. These pictures and number confirm the previous assumptions pretty well. All

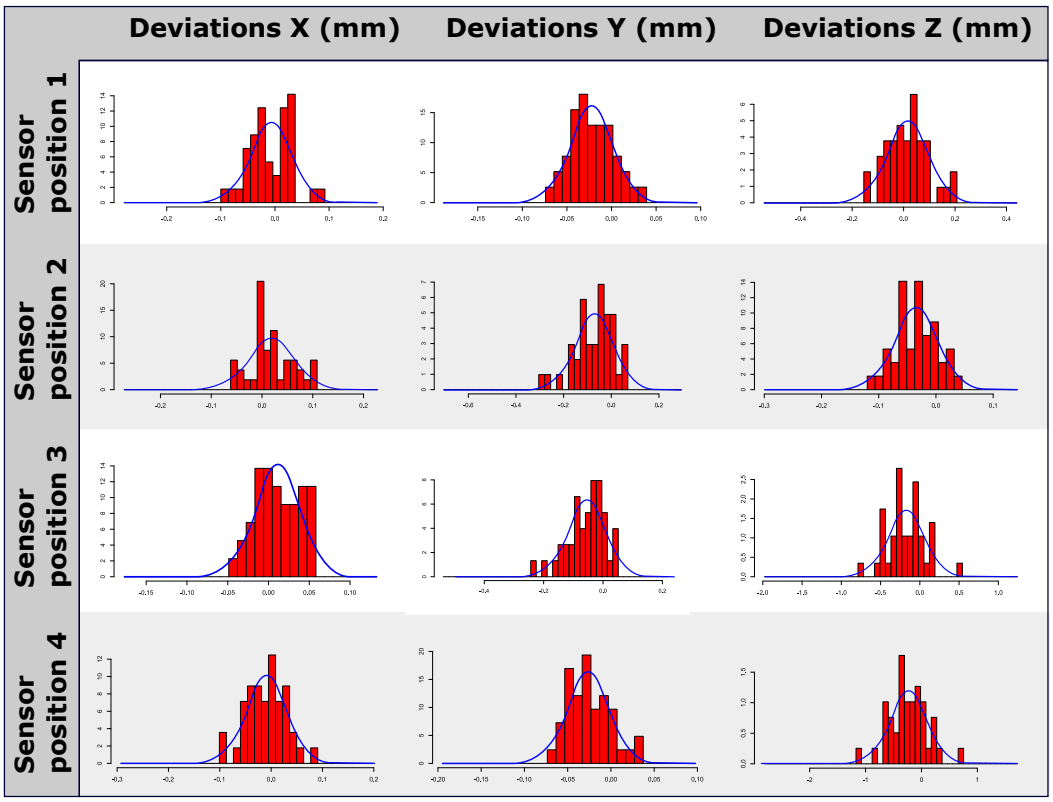

Figure 5.6: Histograms of the deviation after correction of 5mm translation and 0.5°shift.

differences between the first and the second test seem to be of statistical nature due to the small sample size of 20. From the magnitudes the results look equally.

5.2.3 Standard with improved feature detection

For the previous two tests the feature detection was the common standard method called "contour tear off" detection. This very simple method just searches the last contour point from a certain direction. To analyze the improvements for the proposed methods for feature detection *[3.2]* and image stabilization *[3.1]* this third test had been implemented. The setup is equal to the first test, which is similar to the industrial setup, but all the sensor images are filtered and the contour matching had been used for sub-pixel accuracy matching. Table 5.3 presents the results of this previously described test. One can clearly see the stabilization of the feature detection in the sensor images lead to a huge improvement of the results. The mean deviation at each measurement spot after correction are with about 0.01mm about 5 times smaller, as before. Furthermore the 3σ measurement uncertainty converges very clearly to measurement uncertainty denoted for the sensor, which is about $\pm 0.5mm$. Finally image ***(Fig. 5.8)*** present the graphical results of test 3.

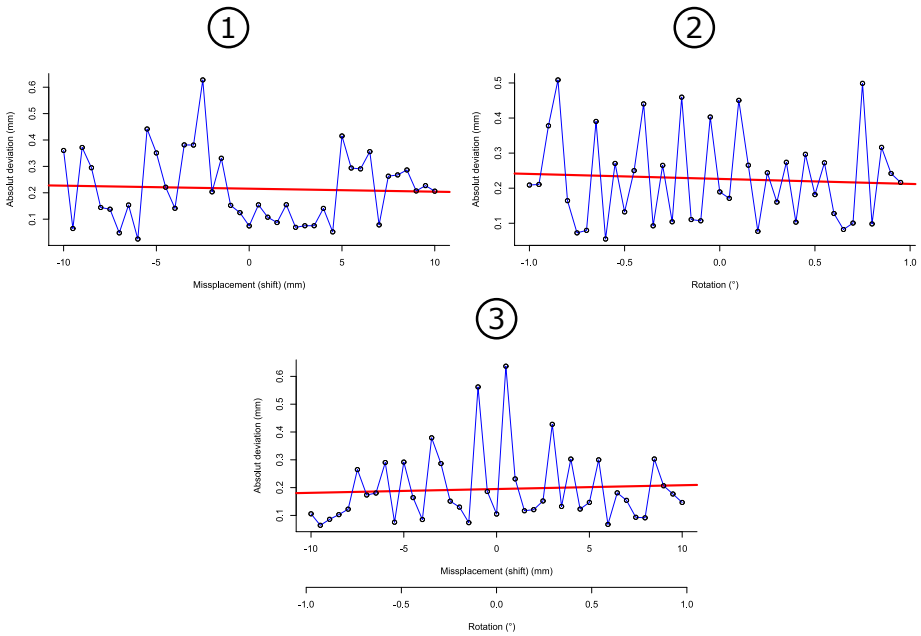

Figure 5.7: Test results for Test 2 with translation only in sub image 1, rotation only in sub image 2 and translation with rotation in sub image 3

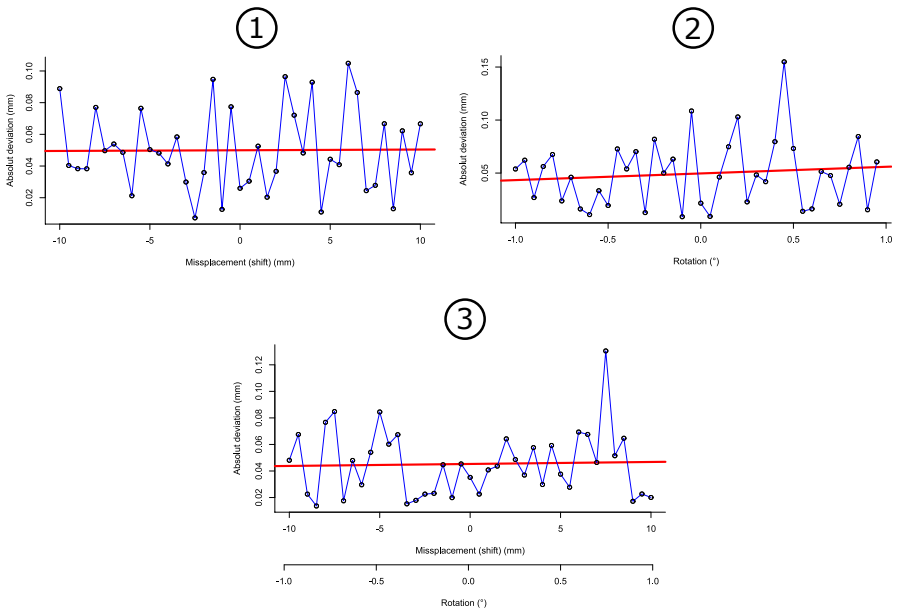

Figure 5.8: Test results for Test 3 with translation only in sub image 1, rotation only in sub image 2 and translation with rotation in sub image 3

Table 5.3: Test results of test 3

	Mean	Min	Max	Var	3σ
Sub Fig. 1	-0,0118 mm	-0,1346 mm	0,0414 mm	0,0007 mm	0,0769 mm
Sub Fig. 2	-0,0107 mm	-0,1942 mm	0,0955 mm	0,0008 mm	0,0864 mm
Sub Fig. 3	-0,0123 mm	-0,1645 mm	0,0375 mm	0,0003 mm	0,0527 mm

5.2.4 Misaligned sensor positions

As mentioned many times before the proposed algorithm assumes, that the motion directions of the robot are aligned with the unobservable degree of freedom of the sensor. Or at least, if a perfect alignment is not possible, one must adjust the weight vector $\underline{w}$ in a way that the deltas, cause by misplacement, will cancel out. Hence it is crucial to have an accurate sensor setup on the measurement target, for being able to guarantee good results. In practice it turned out being a good idea to employ gauges[6] for adjusting the laser. With such tools it is very possible to ensure a good alignment. For the last two test the goal however was to find out how much a bad alignment influences the measurement results. The results of this test can be inspected in table 5.4. Apart from the mean, which is quite similar to the values

Table 5.4: Test results of test 4

	Mean	Min	Max	Var	3σ
Sub Fig. 1	0,1263 mm	-0,8482 mm	1,4765 mm	0,1468 mm	1,1493 mm
Sub Fig. 2	0,0726 mm	-1,8866 mm	1,2093 mm	0,2892 mm	1,6133 mm
Sub Fig. 3	0,0542 mm	-2,0709 mm	0,8042 mm	0,2658 mm	1,5466 mm

before, the 3σ values speak a clear language. The values are literally exploding compared to the test before. This test clearly emphasizes how important it is to have a clean setup especially regarding the rotation around the z-axis of the sensor frame. The surprisingly good mean values are related to the fact that for one direction of misplacement the values increase in positive direction and for the other direction they get negative. Hence in average the values are looking good, the variance however is showing the real deviations. From the theory it is expected for the deviations to increase linearly according to the distance to the zero position (zero shift). Whereby the slope of the linear increasing errors is dependent on the angle by which the sensor is misaligned and the local linearity.

5.2.5 Misaligned sensor positions (not continuous)

Although the previous test already demonstrated how crucial a good alignment is, it was also of interest how much additional error had been caused by trigger inaccuracy within test 4. Therefore the robot has been stopped at each measurement spot within this final

[6]The systems had been installed all over the world and in practice there was not always "professional" good gauge by hand. It commonly happened that the installing engineer just employed plastic cards (like credit cards) which turned out to be perfect for perpendicular alignment due to their straight edges and right angles.

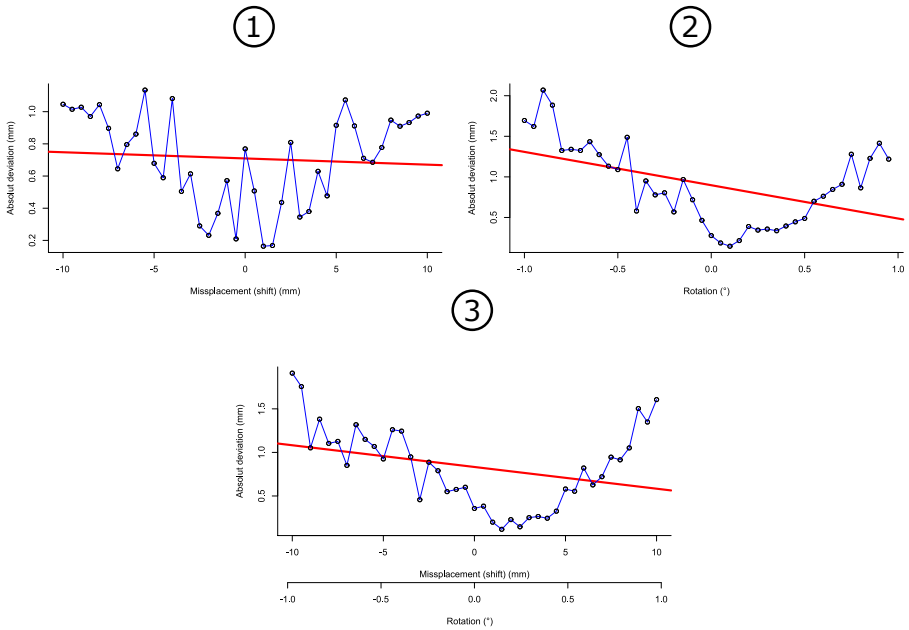

Figure 5.9: Test results for Test 4 with translation only in sub image 1, rotation only in sub image 2 and translation with rotation in sub image 3

experiment. This ensures that the measurements will be taken at the right spots without the trigger jitter, which is, regarding to test 4, prone to high inaccuracies. As expected one can

Table 5.5: Test results for Test 5 with translation only in sub image 1, rotation only in sub image 2 and translation with rotation in sub image 3

	Mean	Min	Max	Var	3σ
Sub Fig. 1	0,0334 mm	-0,8137 mm	0,7679 mm	0,1153 mm	1,0189 mm
Sub Fig. 2	-0,0077 mm	-2,1940 mm	1,2953 mm	0,2577 mm	1,5229 mm
Sub Fig. 3	-0,0223 mm	-2,0024 mm	0,6190 mm	0,2504 mm	1,5012 mm

find slightly smaller values within table 5.10 for the variances and the resulting measurements uncertainties, compared to test 4. The influence of the trigger jitter gets even more evident within the graphs, where once can now see the expected linear inclinations quite clear and less distorted.

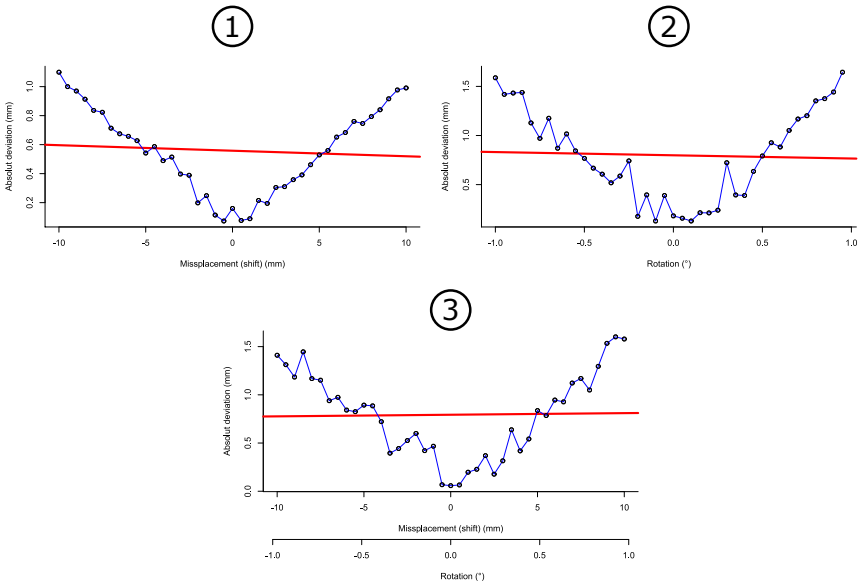

Figure 5.10: Test results for Test 5 with translation only in sub image 1, rotation only in sub image 2 and translation with rotation in sub image 3

Chapter 6

Conclusion

6.1 Outline of contributions

This thesis proposes a new set of methods to employ laser line scanner data for an industrial measurement system. It covers systems for path correction, as well as panel fitting systems. The methods described in chapter *[3.3.1]* concentrate on industrial path correction systems and chapter *[4.2.1]*extends the previously introduced methods to panel fitting.

The first section *[3.1]* of chapter *[3.3.1]*proposes a novel method for filtering the sensor data. This method uses a unique stochastic approach to determine probabilities for candidates of line points. This is due to the issue that common laser line scanners often confuse noise or light reflections as valid line points and include them in the images. By applying the proposed method, one receives a probability for each point being a valid measure and is therefore able to filter by the odds of the points being noise.

Further, as the next logical step, section *[3.2]* proposes a new method to match patterns within the sensor images. The purpose of the pattern matching is to determine the point of interest as a fixed landmark within the sensor image. One is able to safely identify such a landmark because it can be tracked within the images and serve as reference for correction. The proposed method offsets itself form existing methods essentially by improving the bottleneck of detection rate.

And, finally, as the last logical step, section *[3]* concentrates on the algorithm for correcting the robot's trajectory itself. This section outlines a way to calculate a complete transformation from the sensor data. It introduces a mathematical model to determine position and orientation by combining a set of sparse sensor data to a fully qualified correction vector.

Chapter *[4]* focuses on visual servoing. It concentrates on the changes necessary to adjust the methods proposed in the previous chapter *[4.2.1]* from the agile path correction to the more static panel-fitting tasks. Panel fitting usually requires a setup of sensors mounted within fixed positions around the robot's grabber. It was, therefore, necessary to develop a

new kind of calibration, introduced in section *[4.1]*, where the sensors can be calibrated within a single position.

Another difference between panel fitting and the path correction is that panel fitting is a relative measurement. To fit the panel correctly into the frame, one needs to measure both the frame and the panel. After that, we can adjust the panel relative to the frame. Therefore, the algorithm introduced in Section *[3]* needed to be adjusted to perform a relative fitting. This adjustment was introduced in section *[4.2]*. The following publications gather all these contributions:

- Optimization and filtering of the sensor data: *"A MAP estimator based on geometric Brownian motion for sample distances of laser triangulation data"* [71]
- Feature detection within the sensor data:
  1. *"Fast and robust point cloud matching based on EM-ICP prepositioning"*[48]
  2. *"Automatisierung von Fertigungsprozessen großvolumiger Bauteile"*[72]
- Path correction: *"An over-determined path correction algorithm for sparse dimensional measurements"*[4]
- Static laser line sensor calibration: *"A method to determine the extrinsic parameter of laser triangulation sensors, with restricted mobility"*[73]
- Panel fitting algorithm: *"A novel approach for automated car body panel fitting."*[74]

Many of the methods, developed within this work, have also been implemented within the VMT BK-System in parallel and proofed to work within an industrial environment. And, finally, today VMT BK is the de facto standard system for measuring visual sealing beads for most of the premium car producers worldwide.

6.2 Outlook

The methods for path correction proposed in this thesis are concentrating on offline path correction. In offline path correction, the robot first measures and then performs its application.

The next logical step for extension would therefore be an online path correction, where both steps are combined and therefore save more cycle time. However, this approach is strongly dependent on real-time interfaces, like KUKA RSI, which are not available on every robot type yet. And for robots which provide real-time interfaces today, one will suffer with the strong architectural differences between different manufacturers.
For online path correction, the sensor is another critical element. A sensor for an online system needs to measure ahead of the application. That cannot be achieved by mounting a sensor ahead from the point of application. If the contour bends, the sensor needs go around this contour bend before the robot (and the application). This can only be achieved by enriching the sensor with an additional independent degree of freedom or by having different view ports

within the sensor.

Another limitation of the path correction is the assumption of the linear continuation of the contour. This assumption seems to hold in most cases since, as discussed in section *[5.1]*, the error is almost always negligible. But for special spots with a strong curvature, like the wheelhouse for example, one might want to employ different model assumptions.

Another interesting field of research would be the continuous inspection. This inspection would be performed while the sensors are scanning the part for path correction. The difference would be that the sensor collects a first scan which will be provided to the path correction algorithm and after that it starts to scan the contour continuously until the next measuring point. The continuous scans could be used to evaluate if the gaps provide sufficient space for the application. This could be a way to prevent tearing off nozzles in gaps that are too tight. Today, some systems already perform a gap check but apply much simpler evaluations due to performance issues. If the performance of the proposed contour matching of *[3.2]* could be further improved, a substantial improvement of the gap inspection would be possible. Applying a contour detection method once would not only get more accurate results, but would also get the gaps orientation. A decision of the validity of the gap would therefore be much more reliable.

Regarding the calibration, the demand of further development might be to determine a calibration method which gets along with little or no motion, but without employing the camera images. Since, as mentioned in *[4.1]*, there are laser line scanners that do not provide their camera image, this might be an interesting approach.

Appendix A

Linear least square problems

A.1 Linear least square

In general, there are usually two type of errors:

- Systematic errors
- Random errors

The systematic errors are fully predicable and can be approximated by a model equation. There might, however, be situations where models get too complex and it would just not be efficient to model the reality with such a complexity to avoid all systematic errors. An example of such a compromise is the path-correction model, which was presented in chapter *[3]* and simplifies the model equation by assuming the observed contour to progress linearly.

Other errors are random. They are caused by unknown and unpredictable changes. They cannot be modeled or predicted. Those errors can usually be understood as a big number of statistically independent random variables and can therefore be assumed to be approximately normally distributed due to the CLT [1].

This is because whenever a physical quantity gets measured, the measurement gets influenced by numerous disturbances, which overlay the measured quantity with noise. These disturbances may have various distributions, but as the CLT states:

> A sum over a sequence of independent but not necessarily equally distributed random variables, having finite mean and variance, satisfying the Lindeberg Condition, converges to a normal distribution.

The Lindeberg Condition is given in equation ***(A.1)*** and for the discrete case with a given sequence of samples $X_1, X_2, ...$ it is given as in ***(A.2)***. Details on the Lindeberg Condition can be found in [75]. Hence since one expects the resulting error to be a composition of

[1] Central Limit Theorem

random effects all fulfilling Lindeberg Condition, one can also expect the error to be normally distributed.

$$\lim_{n\to\infty} \frac{1}{s^2} \sum_{k=1}^{n} \int_{|x-\mu_k|>\varepsilon s_n} (x-\mu_k)^2 f_k(x)dx = 0 \tag{A.1}$$

$$\lim_{n\to\infty} \frac{1}{s^2} \sum_{k=1}^{n} \mathbb{E}\left[(X_k-\mu)^2 \cdot 1_{|x-\mu_k|>\varepsilon s_n}\right] \tag{A.2}$$

$$where \quad s_n^2 := \sum_{k=1}^{n} \sigma_k^2 \tag{A.3}$$

This allows us to use equation ***(A.4)*** as an assumption of the distribution for the measurements.

$$f(\underline{x}|\underline{\mu},\Sigma) \sim \frac{1}{(2\pi)^{\frac{p}{2}}|\Sigma|^{\frac{1}{2}}} e^{\left(-\frac{1}{2}(\underline{x}-\underline{\mu})^T\Sigma^{-1}(x-\underline{\mu})\right)} \tag{A.4}$$

Where $\underline{x}$ is an $p \times 1$ vector of the set of measurement values X, $\underline{\mu}$ represents the $p \times 1$

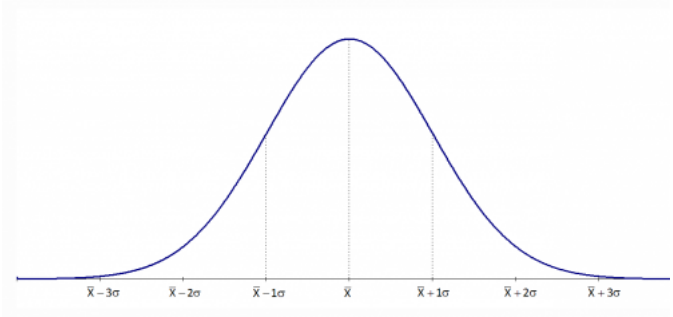

Figure A.1: Normal distribution

vector of mean values and Σ is the $p \times p$ covariance Matrix. Figure ***(Fig. A.1)*** depicts such a normal distribution for the case $p = 1$.

Since, due to CLT, such normal distributed measures are very common, the math to estimate the real, noise less values, is a well know scientific discipline called "regression analysis". The first ideas are dating back to Gauss and Legendre, who were both tying to solve planetary[2] trajectories by applying these ideas. And both were able to predict planetary positions, by employing theses techniques, with incredible accuracy. Furthermore, they achieved their results independent from each other at almost the same time, which lead to a bit of a discussion about who was first. The next sections will give a deeper insight into these ideas and will derive a solution for a general linear case.

[2] Gauss way trying to predict the position of Ceres and Legendre was acutely trying to calculate asteroid positions.

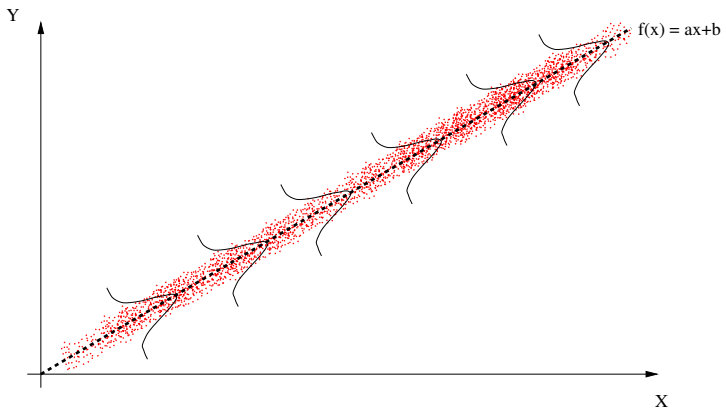

Figure A.2: Normal distributed measurements around a linear function.

The mean value of the normal distribution itself doesn't necessarily need to be a fixed single value. It's also possible to consider some mean function $f_\mu(\underline{x})$.To avoid being too general, in the first step we consider a simple linear function as an example. Figure ***(Fig. A.2)*** shows such a linear mean function *(dotted line)* surrounded by a set of measurements *(red dots)* and their schematic distributions. Since the function is a scalar function, we can simplify the distribution of ***(A.4)*** to a scalar function as well. This is given within equation ***(A.5)***. The vectors and matrices of ***(A.4)*** turn out to be scalar values in this concrete example due to the fact that we only deal with one degree of freedom.

$$f_i(x|f_\mu,\sigma^2) \sim \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_i - f_\mu(x_i))^2}{2\sigma^2}} \tag{A.5}$$

The i within function ***(A.5)*** indicates the i^{th} experiment of n experiments. Or, in other words, one can consider a set of experiments X with the cardinality $|X| = n$ and $x_i \in X$ as one of its elements. As soon as all measurements are iid., we will be able to formulate an expression for the over all density which simply is the product of each measurement density ***(A.6)***.

$$f_X(X|f_\mu,\sigma^2) = \prod_{x_i \in X} f(x_i|f_\mu,\sigma^2) \tag{A.6}$$

Equation ***(A.6)*** represents the probability of all data within the set X for a given parameter set f_μ and σ^2. Since the data is known but the parameters are unknown, we will work out in the following section how to determine an optimal parameter set.

A.1.1 Optimal model parameters

In the previous example, we considered that we have a set of measurements X but we do not know the distributions parameters. Thus, one needs to take a deeper look at the parameter

of ***(A.6)*** which has been the denoted as the variance σ^2 and the mean function as $f_\mu(x)$. Since it is known that for the given example the mean function is a one-dimensional linear function, we can assume it looks like stated in equation ***(A.7)***.

$$f_\mu(x|a,b) = ax + b \tag{A.7}$$

Now we can create a density function by using ***(A.7)*** within the general density function as given in ***(A.6)***. Since the data now is fixed but the parameters of ***(A.6)*** are the variables, we can switch them to get an expression representing the probability of the parameters. This density function is the so-called *likelihood function* ***(A.8)*** and will be denoted by a $\mathcal{L}$. Since the measurements are well known within X, the x can be understood as fixed, the missing parameters a, b and σ^2 are now the variables. If all the assumptions about the probability distribution of those measurements in X and the underlying model are right, finding optimal parameters is straightforward form this point on. The best, or ***optimal***, parameters are those that maximize the probability density of the likelihood function ***(A.9)*** and therefore for the unknown elements.

$$\mathcal{L}(a,b|x) = \prod_{x_i \in X} \frac{1}{(2\pi)^{\frac{1}{2}}\sigma} e^{-\frac{(x_i - f_\mu(x_i|a,b))^2}{2\sigma^2}} \tag{A.8}$$

$$\max_{\forall a,b \in \mathbb{R}} \mathcal{L}(a,b|x) = \max_{\forall x_i \in X} \prod_{x_i \in X} \frac{1}{(2\pi)^{\frac{1}{2}}\sigma} e^{-\frac{(x_i - f_\mu(x_i|a,b))^2}{2\sigma^2}} \tag{A.9}$$

$$= \max_{\forall x_i \in \mathbb{R}} \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{n}{2}} e^{\left(-\frac{\sum_{i=1}^{n}(x_i - f_\mu(x_i|a,b))^2}{2\sigma^2}\right)} \tag{A.10}$$

The product term of ***(A.9)*** can be rewritten into a sum, which is shown in equation ***(A.10)***. In equation ***(A.10)*** one can easily see that, due to the properties of the exponential function, the maximum refers to the minimum of its argument. Therefore, we can formulate the problem as stated in ***(A.13)***.

$$\max_{\forall a,b \in \mathbb{R}} \mathcal{L}(a,b|x) = \max_{\forall x_i \in X} \prod_{x_i \in X} \frac{1}{(2\pi)^{\frac{1}{2}}\sigma} e^{-\frac{(x_i - f_\mu(x_i|a,b))^2}{2\sigma^2}} \tag{A.11}$$

$$= \min_{\forall x_i \in \mathbb{R}} \left(-\frac{\sum_{i=1}^{n}(x_i - f_\mu(x_i|a,b))^2}{2\sigma^2}\right) \tag{A.12}$$

$$= \min_{\forall x_i \in \mathbb{R}} \sum_{i=1}^{n}(x_i - f_\mu(x_i|a,b))^2 \tag{A.13}$$

With formula ***(A.13)***, we have now a clear expression to find the optimal parameters of the given linear function. The model function $f_\mu(x_i|a,b)$ can then be replaced with the expression as given in equation ***(A.7)***. To get the minimum of this example function, one can consider to search for the first derivative of equation ***(A.13)*** and set it to zero. Doing so leads to equation ***(A.14)***. From this point on, methods of standard calculus lead us towards the desired variables. Therefore, we start with the first derivation with respect to b to find the

first expression ***(A.15)***.

$$
\begin{aligned}
& \frac{d}{db}\sum_{i=1}^{n}(y_i-(ax_i+b))^2=0 \\
\Leftrightarrow\quad & \sum_{i=1}^{n}\frac{d}{db}(y_i-ax_i-b)^2=0 \\
\Leftrightarrow\quad & \sum_{i=1}^{n}-2y_i+2ax_i+2b=0 \\
\Leftrightarrow\quad & \sum_{i=1}^{n}2y_i-2ax_i=\sum_{i=1}^{n}2b \\
\Leftrightarrow\quad & \frac{1}{n}\sum_{i=1}^{n}y_i-\frac{1}{n}\sum_{i=1}^{n}ax_i=b \qquad \text{(A.14)} \\
\Leftrightarrow\quad & \bar{y}-a\bar{x}=b \qquad \text{(A.15)}
\end{aligned}
$$

So far, we received an expression for b, so we need to do the same for a, which leads us to equation ***(A.16)***.

$$
\begin{aligned}
& \frac{d}{da}\sum_{i=1}^{n}(y_i-(ax_i+b))^2=0 \\
\Leftrightarrow\quad & \sum_{i=1}^{n}\frac{d}{da}(y_i-ax_i-b)^2=0 \\
\Leftrightarrow\quad & \sum_{i=1}^{n}-2y_ix_i+2ax_i^2+2bx_i=0 \\
\Leftrightarrow\quad & \sum_{i=1}^{n}y_ix_i-\sum_{i=1}^{n}bx_i=\sum_{i=1}^{n}ax_i^2 \\
\Leftrightarrow\quad & \sum_{i=1}^{n}y_ix_i-\sum_{i=1}^{n}\bar{y}x_i+a\sum_{i=1}^{n}\bar{x}x_i=\sum_{i=1}^{n}ax_i^2 \\
\Leftrightarrow\quad & \sum_{i=1}^{n}y_ix_i-\sum_{i=1}^{n}\bar{y}x_i=a(\sum_{i=1}^{n}x_i^2-\sum_{i=1}^{n}\bar{x}x_i) \\
\Leftrightarrow\quad & \frac{\sum_{i=1}^{n}(y_ix_i-\bar{y}x_i)}{(\sum_{i=1}^{n}x_i^2-\sum_{i=1}^{n}\bar{x}x_i)}=\frac{\sum_{i=1}^{n}y_ix_i-\frac{1}{n}\sum_{i=1}^{n}y_i\sum_{i=1}^{n}x_i}{\sum_{i=1}^{n}x_i^2-\frac{1}{n}\left(\sum_{i=1}^{n}x_i\right)^2} \\
=\quad & \frac{\sum_{i=1}^{n}(y_ix_i-\bar{y}x_i)}{(\sum_{i=1}^{n}x_i^2-\sum_{i=1}^{n}\bar{x}x_i)}=\frac{Cov\left[x,y\right]}{Var\left[x\right]}=a \qquad \text{(A.16)}
\end{aligned}
$$

With formula ***(A.16)***, we derive a very compact expression for the model parameter a, which only relies to the variance and the covariance of the given measurements. Generally, it is possible to formulate a formalism in order to find the best-fit parameters for arbitrary linear and multivariate model functions. This will be discussed in the following section.

A.1.2 Generalization for the linear case

For the generalization of the given equation in *[A.1.1]*, we first consider a general linear function, as in ***(A.17)***, where the x_j's indicate the known measurement positions, y represents the measurement itself, and the β_j's are the unknown model parameters.

$$f(x_1, x_2, ..., x_m) = x_1\beta_1 + x_2\beta_2 + ... + x_n\beta_n = \sum_{j=1}^{m} x_j\beta_j = y \tag{A.17}$$

If we take, for example, the simple line model ***(A.7)*** of section *[A.1.1]*, and transfer it to this scheme, it will result in equation ***(A.18)***.

$$f(x) = \beta_2 x_1 + \beta_1 \cdot 1 \tag{A.18}$$

Each measurement can be seen as an independent observation of the given model ***(A.17)***. Therefore, we introduce an additional index i for the observed values, to indicate the i^{th} observation of all n observations. Since the observed variables are x and y, we need to change x_j to $x_{i,j}$, called the j-th position in the i-th observation and y to y_i the result of the i^{th} observation. Doing so changes the simple model of ***(A.13)*** in section *[A.1.1]* to equation ***(A.19)***. Here m denotes the number of dimensions and therefore also the number of unknowns and n is the number of observations.

$$\sum_{i=0}^{n} \left(y_i - \sum_{j=0}^{m} x_j\beta_{i,j} \right)^2 \rightarrow min \qquad \forall i \in n \wedge \ m \in \mathbb{N} \tag{A.19}$$

Similar to what we did in section *[A.1.1]* we have to find the partial derivative for each unknown element x_j in the general equation in order to find the minimum of the given expression ***(A.19)***. This results into n equations, which need to be solved for their unknowns.

$$\begin{array}{rcl} \frac{\delta}{\delta\beta_1} \sum_{i=0}^{n} \left(y_i - \sum_{j=0}^{m} \beta_j x_{i,j} \right)^2 & = & 0 \\ \frac{\delta}{\delta\beta_2} \sum_{i=0}^{n} \left(y_i - \sum_{j=0}^{m} \beta_j x_{i,j} \right)^2 & = & 0 \\ \vdots & & \vdots \\ \frac{\delta}{\delta\beta_m} \sum_{i=0}^{n} \left(y_i - \sum_{j=0}^{m} \beta_j x_{i,j} \right)^2 & = & 0 \end{array} \tag{A.20}$$

$$\begin{aligned}
\sum_{i=0}^{n}\left[2\left(y_i-\sum_{j=0}^{m}\beta_j x_{i,j}\right)\cdot x_{i,1}\right] &= 0\\
\sum_{i=0}^{n}\left[2\left(y_i-\sum_{j=0}^{m}\beta_j x_{i,j}\right)\cdot x_{i,2}\right] &= 0\\
\vdots &\quad\vdots\\
\sum_{i=0}^{n}\left[2\left(y_i-\sum_{j=0}^{m}\beta_j x_{i,j}\right)\cdot x_{i,m}\right] &= 0
\end{aligned} \tag{A.21}$$

$$\begin{aligned}
\sum_{i=0}^{n} y_i x_{i,1}-\beta_1\sum_{i=0}^{n}x_{i,1}^2+\beta_2\sum_{i=0}^{n}x_{i,2}x_{i,1}+\cdots+\beta_m\sum_{i=0}^{n}x_{i,m}x_{i,1} &= 0\\
\sum_{i=0}^{n} y_i x_{i,2}-\beta_1\sum_{i=0}^{n}x_{i,1}x_{i,2}+\beta_2\sum_{i=0}^{n}x_{i,2}^2+\cdots+\beta_m\sum_{i=0}^{n}x_{i,m}x_{i,2} &= 0\\
&\vdots\\
\sum_{i=0}^{n} y_i x_{i,m}-\beta_1\sum_{i=0}^{n}x_{i,1}x_{i,m}+\beta_2\sum_{i=0}^{n}x_{i,2}x_{i,m}+\cdots+\beta_m\sum_{i=0}^{n}x_{i,m}^2 &= 0
\end{aligned} \tag{A.22}$$

With equation ***(A.22)*** we now have a general expression for arbitrary linear equations. We need to rearrange this equation to bring it into the shape of a linear matrix equation.

$$\begin{aligned}
\sum_{i=0}^{n} y_i x_{i,1} &= \beta_1\sum_{i=0}^{n}x_{i,1}^2+\beta_2\sum_{i=0}^{n}xi,2x_{i,1}+\cdots+\beta_m\sum_{i=0}^{n}x_{i,m}x_{i,1}\\
\sum_{i=0}^{n} y_i x_{i,2} &= \beta_1\sum_{i=0}^{n}x_{i,1}x_{i,2}+\beta_2\sum_{i=0}^{n}x_{i,2}^2+\cdots+\beta_m\sum_{i=0}^{n}x_{i,m}x_{i,2}\\
\vdots &\quad\vdots\\
\sum_{i=0}^{n} y_i x_{i,m} &= \beta_1\sum_{i=0}^{n}x_{i,1}x_{i,m}+\beta_2\sum_{i=0}^{n}x_{i,2}x_{i,m}+\cdots+\beta_m\sum_{i=0}^{n}x_{i,m}^2
\end{aligned} \tag{A.23}$$

With the representation we have in equation ***(A.23)*** we now can formulate it as a matrix equation, which is done in equation ***(A.24)***.

$$\begin{pmatrix}
\sum_{i=0}^{n}x_{i,1}^2 & \sum_{i=0}^{n}x_{i,2}x_{i,1} & \cdots & \sum_{i=0}^{n}x_{i,m}x_{i,1}\\
\sum_{i=0}^{n}x_{i,1}x_{i,2} & \sum_{i=0}^{n}x_{i,2}^2 & \cdots & \sum_{i=0}^{n}x_{i,m}x_{i,2}\\
\vdots & \vdots & \ddots & \vdots\\
\sum_{i=0}^{n}x_{i,1}x_{i,m} & \sum_{i=0}^{n}x_{i,2}x_{i,m} & \cdots & \sum_{i=0}^{n}x_{i,m}^2
\end{pmatrix}\cdot\begin{pmatrix}\beta_1\\ \beta_2\\ \vdots\\ \beta_m\end{pmatrix}=\begin{pmatrix}
\sum_{j=0}^{m}y_i x_{1,j}\\
\sum_{j=0}^{m}y_i x_{2,j}\\
\vdots\\
\sum_{j=0}^{m}y_i x_{n,m}
\end{pmatrix} \tag{A.24}$$

The left hand matrix in ***(A.24)*** is of $m \times m$ dimension. If the original problem was not singular, this matrix should also be of rank m. Therefore it is invertible, which allows us to the next step, where we rearrange equation ***(A.24)*** to ***(A.25)***. With equation ***(A.25)*** we now

have an expression for the desired model parameter.

$$\begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{n} x_{i,1}^2 & \sum_{i=0}^{n} x_{i,2}x_{i,1} & \cdots & \sum_{i=0}^{n} x_{i,m}x_{i,1} \\ \sum_{i=0}^{n} x_{i,1}x_{i,2} & \sum_{i=0}^{n} x_{i,2}^2 & \cdots & \sum_{i=0}^{n} x_{i,m}x_{i,2} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=0}^{n} x_{i,1}x_{i,m} & \sum_{i=0}^{n} x_{i,2}x_{i,m} & \cdots & \sum_{i=0}^{n} x_{i,m}^2 \end{pmatrix}^{-1} \cdot \begin{pmatrix} \sum_{j=0}^{m} y_i x_{1,j} \\ \sum_{j=0}^{m} y_i x_{2,j} \\ \vdots \\ \sum_{j=0}^{m} y_i x_{n,m} \end{pmatrix} \tag{A.25}$$

The matrix equation of ***(A.25)*** leads us to the optimal parameter set of the given arbitrary model ***(A.17)***. Now we can go back to the initial model equation as given in ***(A.17)***. We define 3 matrices where each row represents one observation. All positions of each observation in each degree of freedom $x_{i,j}$ are stored in the Matrix A, the so-called design matrix. The design matrix A is of the dimension $n \times m$, where n is the number of observation and m is the dimension of the model. All measurement results will be coupled up in the result vector $\underline{y}$. Finally, we create a vector with all unknown parameter elements β_j called $\underline{\beta}$ ***(A.27)***.

$$A = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{pmatrix} \ , \ \underline{\beta} = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{pmatrix} \ , \ \underline{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \tag{A.26}$$

If one now sets up the equation $A\underline{\beta} = \underline{y}$ it will directly result into the original definition of equation ***(A.17)***, which is drafted in equation ***(A.27)***.

$$A\underline{\beta} = \begin{pmatrix} x_{1,1} \cdot \beta_1 + x_{1,2} \cdot \beta_2 + \cdots + x_{1,m} \cdot \beta_m \\ x_{2,1} \cdot \beta_1 + x_{2,2} \cdot \beta_2 + \cdots + x_{2,m} \cdot \beta_m \\ \vdots \\ x_{n,1} \cdot \beta_1 + x_{n,2} \cdot \beta_2 + \cdots + x_{n,m} \cdot \beta_m \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} = \underline{y} \tag{A.27}$$

With ***(A.27)*** we now have a general formulation for the linear system as a Matrix. To find the optimal vector, we have to bring it into the shape as stated in equation ***(A.25)***. Therefore, we take a deeper look at the design matrix A. Equation ***(A.28)*** shows that the product of the transposed of A with A results exactly into the left-hand matrix stated in equation ***(A.25)***.

$$A^T A = \begin{pmatrix} \sum_{i=0}^{n} x_{i,1}^2 & \sum_{i=0}^{n} x_{i,2}x_{i,1} & \cdots & \sum_{i=0}^{n} x_{i,m}x_{i,1} \\ \sum_{i=0}^{n} x_{i,1}x_{i,2} & \sum_{i=0}^{n} x_{i,2}^2 & \cdots & \sum_{i=0}^{n} x_{i,m}x_{i,2} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=0}^{n} x_{i,1}x_{i,m} & \sum_{i=0}^{n} x_{i,2}x_{i,m} & \cdots & \sum_{i=0}^{n} x_{i,m}^2 \end{pmatrix} \tag{A.28}$$

We can also achieve the desired result vector of ***(A.25)*** if we multiply the transposed of A with the vector of measurement samples $\underline{y}$, as stated in ***(A.29)***.

$$A^T \underline{y} = \begin{pmatrix} \sum_{j=0}^{m} y_i x_{1,j} \\ \sum_{j=0}^{m} y_i x_{2,j} \\ \vdots \\ \sum_{j=0}^{m} y_i x_{n,m} \end{pmatrix} \tag{A.29}$$

This leads us to a general equation for the estimator of the parameter vector $\underline{\beta}$ as given in ***(A.30)***.

$$\underline{\beta} = \left(A^T \cdot A\right)^{-1} \cdot A^T \cdot \underline{y} \tag{A.30}$$

$$= A^{\dagger} \cdot \underline{y} \tag{A.31}$$

The estimator of ***(A.30)*** is often referred to as the *pseudo inverse* or the *Moore-Penrose inverse*. It's usually marked with a dagger symbol† ***(A.31)*** or sometimes by a simple plus symbol $+$.

A.2 Homogeneous least squares problem

In many cases, the linear least square solution is not the desired optimal solution, since there might also be side constraints that have to be met by a valid result. The classic example is the rotation matrix. It is not enough to find a matrix by projecting a set of points ideally to another set of points. In order to have a correct rotation matrix, it is also necessary that each column vector of this rotation matrix is perpendicular to all other column vectors and their length must be 1. So just tearing down all elements into a 9×1 vector and setting up an equation of the style $\mathbf{A}b = c$, which could be solved as described in appendix *[A.1]*, is not sufficient.

In fact, finding a closed form solution for constrained problems can be quite a challenge and will also be impossible for multiple problems. In many situations, the only possibility might be to apply non-linear methods, as described in appendix *[B]*. Nevertheless, if the instant problem is of the form $\|\mathbf{A}h\| = \mathbf{0}$ where h is the $n \times 1$ unknown vector, with the constraint $\|h\| = 1$, there is a straightforward solution for this problem.

Since we are searching the least square minimum of the given problem, we have to get the first derivative of the underlying equation in order to search for the extrema. We also have to apply the method of Lagrangian multipliers, since this is a constrained problem. Hence the additional constraint $\|h\| = 1$ also needs to be squared and rearranged to 0. Thereby it can be added as the subject function $g(h) = 0$ into the Lagrangian expression. The squared subject function is given in equation ***(A.33)***

$$\|h\| = 1 \tag{A.32}$$

$$\Leftrightarrow 1 - h^T h = 0 \tag{A.33}$$

The expression $f(h) = \|\mathbf{A}h\|$ also needs to be squared ***(A.34)***. And after all this, both need to be combined into the Lagrangian expression ***(A.35)*** and finally we have to take the derivative of ***(A.35)*** with respect to h in order to find the minimum ***(A.36)***.

$$\|\mathbf{A}h\| \Rightarrow h^T A^T A h \tag{A.34}$$
$$\Rightarrow h^T A^T A h + \lambda(1 - h^T h) = 0 \tag{A.35}$$
$$\frac{\delta}{\delta h}\left(h^T A^T A h + \lambda(1 - h^T h)\right) = 0 \tag{A.36}$$

In equation ***(A.37)*** we have the first derivative of equation ***(A.36)*** with respect to h, where h can be factored out resulting into equation ***(A.38)***.

$$2\mathbf{A}^T\mathbf{A}h - 2\lambda h = 0 \tag{A.37}$$
$$(\mathbf{A}^T\mathbf{A} - \lambda\mathbf{I})h = 0 \tag{A.38}$$

Equation ***(A.38)*** is called the eigenvalue equation of $\mathbf{A}^T\mathbf{A}$, h is called the **eigenvector** and λ is the **eigenvalue**. The eigenvector is a vector h whose direction will not be changed by the underlying linear transformation. Only the length changes by the factor λ, which is called the eigenvalue. Hence, $\|h\|$ is allays 1 and λ is always the length of h after transforming it with $\mathbf{A}^T\mathbf{A}$, equation ***(A.39)***.

$$\mathbf{A}^T\mathbf{A}h = \lambda h \tag{A.39}$$

Due to the initial equation $\|\mathbf{A}h\|$ it follows that the square error for a given h is $h^T\mathbf{A}^T\mathbf{A}h$ and with the result form ***(A.37)***, we know that the least square error can be rewritten to $h^T\lambda h$.

$$\epsilon = h^T\mathbf{A}^T\mathbf{A}h \tag{A.40}$$
$$\Leftrightarrow h^T\lambda h \tag{A.41}$$

Finally, in equation ***(A.41)***, it is obvious that the smallest eigenvalue is leading to the minimal error ϵ, as well as the biggest eigenvalue is giving the maximum of $\|\mathbf{A}h\|$.
The support for SVD decompositions is very good within many numerical computation libraries and the algorithms for SVD decomposition provide high numerical stability. Hence, it is very desirable to solve the above described problem of $\|\mathbf{A}h\| = \mathbf{0}$ with the given constraint $\|h\| = 1$. Luckily, the derivation of the solution is quite similar to the eigenvector solution described above.

Any matrix can be decomposed by the singular value decomposition. Decomposing a matrix $\mathbf{A}$ results in 3 matrices, $\mathbf{U}$, $\mathbf{\Sigma}$ and $\mathbf{V}^T$ for which the following equation ***(A.42)*** is valid:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{A.42}$$
$$\tag{A.43}$$

If $\mathbf{A}$ has dimension $m \times n$ $\mathbf{U}$ will be of $m \times m$,$\mathbf{V}^T$ will be of $n \times n$ and $\mathbf{\Sigma}$ will be of $m \times n$. The matrix $\mathbf{U}$ is called the left-singular vectors and $\mathbf{V}^T$ the right-singular vectors. Both, $\mathbf{U}$ and $\mathbf{V}^T$ are orthonormal, which means that their components are perpendicular and the norm of the row and column vectors is always 1. That said, it must be clear that the product

between a vector and an orthonormal vector must have the same norm as the equation of the vector itself ***(A.44)***.

$$\left\|\mathbf{V}^T h\right\| = \|h\| \tag{A.44}$$

Hence equation ***(A.45)*** must follow, since $\mathbf{\Sigma V}^T h$ results into a vector.

$$\left\|\mathbf{U\Sigma V}^T h\right\| = \left\|\mathbf{U}\left(\mathbf{\Sigma V}^T h\right)\right\| = \left\|\mathbf{\Sigma V}^T h\right\| \tag{A.45}$$

Finally, the remaining matrix $\mathbf{\Sigma}$, which is called the matrix of singular values, contains all the singular values in descending order. Having this knowledge, we once again take a look at the original problem, of $\|\mathbf{A}h\| = \mathbf{0}$ with $\|h\| = 1$. From equations ***(A.42)*** and ***(A.45)***, we know that equation ***(A.46)*** must follow.

$$\|\mathbf{A}h\| = \left\|\mathbf{U\Sigma V}^T h\right\| = \left\|\mathbf{\Sigma V}^T h\right\| \tag{A.46}$$

We also know that $\left\|\mathbf{V}^T h\right\| = \|h\|$ (***(A.44)***) and that this, given the original constraint, must be 1. If we now substitute $x = \mathbf{V}^T h$ we achieve equation ***(A.47)*** to minimize with subject to ***(A.48)***.

$$min\,\|\mathbf{\Sigma}x\| \tag{A.47}$$

$$\|x\| = 1 \tag{A.48}$$

Recalling that $\mathbf{\Sigma}$ is a diagonal matrix containing the singular values in descending order the minimum must be the last singular value and x therefore must be $(0 \ldots 0, 1)$. With the given substitution for x, it therefore follows that the searched minimum must be the last column of $\mathbf{V}$ due to equation ***(A.50)***.

$$x = \mathbf{V}^T h \tag{A.49}$$

$$\Leftrightarrow \quad h = \mathbf{V}x \tag{A.50}$$

Appendix B

Non linear solving

B.1 Non linear minimisation

Not every mathematical problem has an analytical closed form solution. For multiple reasons, like non linearity or complex constraints an iterative algorithm might be the only way to find a solution. For such kind of iteration schemes one usually needs to supply an error function, which calculates the deviation between the results for the current[1] parameter vector and the desired target properties. Although such functions are usually very straight forward and hence very simple to provide, it is not always preferable to solve all problems iteratively. This is due to the fact that an iterative algorithm usually is considerably slower than a direct calculation, but most important, it might get stuck within a local minimum, or even diverge far away form the result. Therefore it might be necessary to provide an initial guess, from where on the iteration starts. If this guess is far away from potential local minima it might guarantee that the iteration will succeed. This initial guess is called a start value, but again getting these start values is also not always easy. The following sections will provide a general overview of common iterative algorithms to solve non linear minimisation problems. Basically there are two algorithms discussed, first the gradient descent algorithm and second the Gauss-Newton algorithm. Both have their strength and weaknesses, therefore a third algorithm, called Levenberg-Marquardt algorithm, will be introduces which combines both methods to a very efficient algorithm for solving non linear least square problems like given in equation ***(4.28)***.

B.2 Gradient descent algorithm

The gradient descent algorithm is a very simple and straight forward iterative method to find local extrema in differentiable functions. Sometimes it is referred to as the steepest descend algorithm, as well. This algorithm is, like the name implies, based on the gradient of the given objective function. The algorithm itself is based on the observation, that for a continuous function at some point $\underline{x}^{(n)}$ the negative gradient points into the descending direction, at

[1] Within the iteration step.

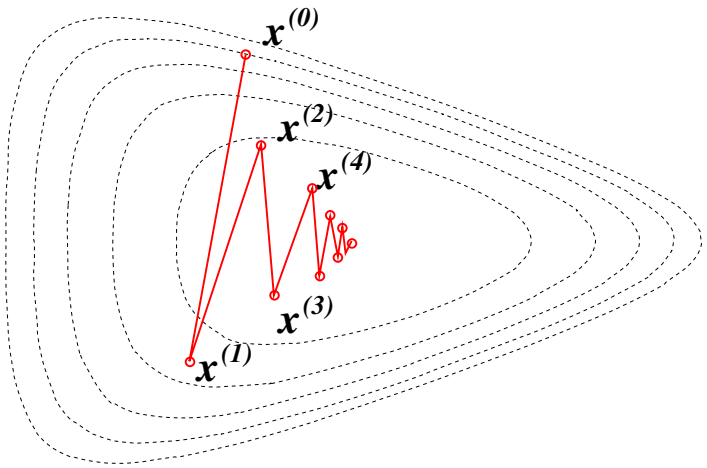

Figure B.1: Illustration of "ping pong"-effect.

least within a arbitrarily small region around $\underline{x}^{(n)}$. Therefore one only has to determine the gradient at the position $\underline{x}^{(n)}$ of the objective function $f(\underline{x}^{(n)})$ and a step width γ for which the objective function decreases. Equation ***(B.1)*** shows such an iteration step at n.

$$\underline{x}^{(n+1)} = \underline{x}^{(n)} - \gamma \nabla f(\underline{x}^{(n)}) \tag{B.1}$$

The function $\nabla f(\underline{x}^{(n)})$ is the derivative of the objective function $f(\underline{x}^{(n)})$ and therefore points in ascending direction. And finally by subtracting it, one moves in the the descending direction. This derivative is called the total derivative and stated in equation ***(B.2)*** where k denotes the cardinality of the parameter vector $\underline{x}$ and $\underline{e}_i$ are the unit vectors in direction of the variables $\underline{x}_i$.

$$\nabla f(\underline{x}) = \sum_{i=1}^{k} \underline{e}_i \frac{\delta f}{\delta x_i} dx_i \tag{B.2}$$

It can be shown, that the convergence speed for the gradient descent algorithm is only linear and depending on the estimated step width γ. Most of the time it gets even worse the closer one iterates towards the minimum. This is, because with a badly chosen step width γ the algorithm tends to descend by "ping ponging" downwards the closer one iterates downhill torwards the minimum. This "ping pong"-effect is illustrated in figure ***(Fig. ??)***. Thus a poor chosen γ-factor can drastically slow down the algorithm the closer one iterates towards the minimum. Due to the importance of choosing a good γ for each step, there exist a vast number of different methods to determine it. To get a better understanding of the problem we take a deeper look at the behavior of the objective function in descent direction around the arbitrary position $\underline{x}^{(n)}$. Therefore we introduce a new function as given in equation ***(B.3)***

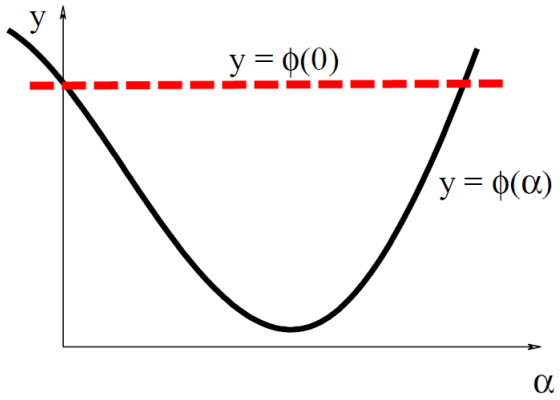

Figure B.2: Illustrates the behavior of $\phi^{(n)}(\gamma)$.

which represents the objective functions behavior in dependence to the step width γ at the current position $\underline{x}^{(n)}$.

$$\phi^{(n)}(\gamma) = f\left(\underline{x}^{(n)} + \gamma \nabla f\left(\underline{x}^{(n)}\right)\right) \tag{B.3}$$

A possible trend for such function $\phi^{(n)}(\gamma)$ is also illustrated in figure ***(Fig. ??)***. However the most fundamental demand for any method determining the step width γ at any position $\underline{x}^{(n)}$ is to fulfill the constraint given as equation ***(B.4)***.

$$\phi^{(n)}(\gamma) < \phi^{(n)}(0) \tag{B.4}$$

This constraint is also denoted by the red line in figure ***(Fig. ??)***. Any point underneath this limit would be valid for a next update step because it assures the desired descending behavior. If we define the set $\Gamma^{(n)}$ as a set of all possible arguments γ for $\phi^{(n)}(\gamma)$ at position $\underline{x}^{(n)}$ fulfilling the given constraint ***(B.4)***, we can define the best possible stepsize $\widehat{\gamma}$ for the actual iteration as stated in ***(B.5)***.

$$\widehat{\gamma} := \min_{\forall \gamma \in \Gamma^{(n)}} (\phi(\gamma)) \tag{B.5}$$

The method to determine $\widehat{\gamma}$ is called the exact line search. The exact line search would always result into the optimal parameter for each iteration step and assure the fastest possible convergence of gradient descent. But on the other hand doing the exact line search is demanding some kind of nonlinear solving process itself and would therefore be inefficient to do it in each step of iteration. Since it is not important to find the exact $\widehat{\gamma}$ but a more or less good approximation there exist a lot of fast methods ought to find an approximation for the exact line search, usually called lossy line search. One of the most common methods is to simply use a constant factor for γ. This very simple method usually is not advisable able because it is not assured that the descending constraint ***(B.4)*** is true for every step and this can result into divergence. Another very popular method is to use a fixed factor ν by which

one can divide γ if the constraint ***(B.4)*** is not true for the current γ. Which means that the step width will successively be decreased.
The method to determine $\widehat{\gamma}$ is called the exact line search. The exact line search would always result into the optimal parameter for each iteration step and assure the fastest possible convergence of gradient descent. But on the other hand doing the exact line search is demanding some kind of nonlinear solving process itself and would therefore be inefficient to do it in each step of iteration. Since it is not important to find the exact $\widehat{\gamma}$ but a more or less good approximation there exist a lot of fast methods ought to find an approximation for the exact line search, usually called lossy line search. One of the most common methods is to simply use a constant factor for γ. This very simple method usually is not advise able because it is not assured that the descending constraint ***(B.4)*** is true for every step and this can result into divergence. An other very popular method is to use a fixed factor ν by which one can divide γ if the constraint ***(B.4)*** is not true for the actual γ. Which means that the step width will successively be decreased if it was not sufficient.
To start the search for the minimum of the objective function one has to choose a start position called $x^{(0)}$. As previously discussed the algorithm iterates from this start point $x^{(0)}$ on towards the descent direction given by the negative gradient $-\nabla f(\underline{x})$ with the step width γ until the stop criterion is fulfilled. This yields to equation ***(B.6)*** for determining the position $\underline{x}^{(n+1)}$ of the upcomming step.

$$\underline{x}^{(n+1)} = \underline{x}^{(n)} - \gamma^{(n)} \nabla f(\underline{x}^{(n)}) \tag{B.6}$$

B.2.1 Gauss-Newton algorithm

The Gauss-Newton algorithm is a specialization of the Newton algorithm to find local minima within least square problems. It's specialization concerns the cubic nature of its objective function. Based on this assumption one will be able to replace the calculation of the Hessian matrix with an approximation, which saves computational complexity and increases numerical stability. But first we take a deeper look at the more general Newton algorithm to get a deeper insight into the basics.
The basic idea for the Newton algorithm is to get an approximation for the objective function $f(x)$ by developing the second order Taylor expansion and setting it's derivative to zero. For a multivariate function this Taylor expansion will result in equation ***(B.7)***.

$$f(\underline{x} + \underline{\phi}) \approx f(\underline{x}) + \underline{\phi}^T J_f(\underline{x}) + \frac{1}{2} \underline{\phi}^T H_f(\underline{x}) \underline{\phi} \tag{B.7}$$

With $J_f(\underline{x})$ as the Jacobian matrix, given as in equation ***(B.8)*** and $H_f(\underline{x})$ as the Hessian matrix, given as in ***(B.9)***.

$$J_f(\underline{x}) = \frac{\delta f(\underline{x})}{\delta \underline{x}} = \begin{pmatrix} \frac{\delta f_1}{\delta x_1} & \frac{\delta f_1}{\delta x_2} & \cdots & \frac{\delta f_1}{\delta x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\delta f_m}{\delta x_1} & \frac{\delta f_m}{\delta x_2} & \cdots & \frac{\delta f_m}{\delta x_n} \end{pmatrix} \tag{B.8}$$

$$H_f(\underline{x}) = \left(\frac{\delta^2 f(\underline{x})}{\delta x_i \delta x_j}\right)_{i,j=1,\ldots,n} = \begin{pmatrix} \frac{\delta^2 f_1}{\delta x_1 \delta x_1} & \frac{\delta^2 f_1}{\delta x_1 \delta x_2} & \cdots & \frac{\delta^2 f_1}{\delta x_1 \delta x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\delta^2 f_m}{\delta x_n \delta x_1} & \frac{\delta^2 f_m}{\delta x_n \delta x_2} & \cdots & \frac{\delta^2 f_m}{\delta x_n \delta x_n} \end{pmatrix} \tag{B.9}$$

Since the Taylor expansion is only an approximation of $f(\underline{x}+\underline{\phi})$ the resulting vector is not the result of the problem, but at least a more or less close approximation. Therefore one needs to repeat this step until the desired abortion criterion is fulfilled. The equations ***(B.12)*** and ***(B.13)*** are showing how to update the function parameter vector $\underline{x}$ for the n-th step of iteration. Calculating the derivative of equation ***(B.7)*** with respect to $\underline{\phi}$ and setting it to zero leads to ***(B.11)***.

$$\frac{d}{d\underline{\phi}}\left(f(\underline{x}^{(n)}) + \underline{\phi}^T J_f(\underline{x}^{(n)}) + \tfrac{1}{2}\underline{\phi}^T H_f(\underline{x}^{(n)})\underline{\phi}\right) \tag{B.10}$$

$$= \quad J_f(\underline{x}^{(n)}) + H_f(\underline{x}^{(n)})\underline{\phi} = 0 \tag{B.11}$$

$$\Leftrightarrow \quad -\underline{\phi} = H_f(\underline{x}^{(n)})^{-1} \cdot J_f(\underline{x}^{(n)}) \tag{B.12}$$

$$\Rightarrow \quad \underline{x}^{(n+1)} = \underline{x}^{(n)} + \underline{\phi} \tag{B.13}$$

Now we can take a deeper look at the differences between the ordinary Newton algorithm and the Gauss-Newton algorithm. The Gauss-Newton algorithm is a specialization for least square problems. Since for least square problems the objective function usually is based on the square over the L2-norm of the differences between some measurements y_i and a model function $f_M(\underline{x})$ which is usually called the residual function ***(B.14)***.

$$r_i^2 = \|y_i - f_M(x_i)\|_2^2 \tag{B.14}$$

Therefore we can consider the objective function to be of the shape $f(\underline{x}) = r(\underline{x})^2$. This extra knowledge provides some benefits with respect to its derivative. Thus one can rewrite the Jacobian into equation ***(B.15)***. With the extra knowledge one can also rewrite the expression for the Hessian matrix. With the help of the product rule and the fact that the derivation of the Jacobian leads to the Hessian we can simplify equation ***(B.9)*** to equation ***(B.16)***.

$$J_f = \begin{pmatrix} 2r_1\frac{\delta r_1}{\delta x_1} & 2r_1\frac{\delta r_1}{\delta x_2} & \cdots & 2r_1\frac{\delta r_1}{\delta x_n} \\ \vdots & \vdots & \ddots & \vdots \\ 2r_m\frac{\delta r_m}{\delta x_1} & 2r_m\frac{\delta r_m}{\delta x_2} & \cdots & 2r_m\frac{\delta r_m}{\delta x_n} \end{pmatrix} = 2J_r r(\underline{x}) \tag{B.15}$$

$$H_f = 2\left(J_r^T J_r + H_r r(\underline{x})\right) \tag{B.16}$$

$$\approx 2J_r^T J_r \tag{B.17}$$

Usually $J_r^T J_r$ is the dominating therm and therefore equation ***(B.17)*** holds for a good approximation of the Hessian H_f. Thus the simplification in the Gauss-Newton algorithm is to replace the Hessian and Jacobian for $f(\underline{x})$ in ***(B.12)*** with the results of ***(B.17)*** and ***(B.15)***. This leads to an expression for each iteration step as given in equation ***(B.18)***.

$$\underline{x}^{(n+1)} = \underline{x}^{(n)} - \left(J_r^T J_r\right)^{-1} \cdot J_r r(\underline{x}) \tag{B.18}$$

Equation ***(B.18)*** yields to less computational complexity and more numerical stability. The rate of convergence of the Gauss-Newton algorithm can approach quadratic time complexity close to a local minimum, which makes it more efficient than the gradient descent algorithm. Convergence is however not guaranteed and significant worse than the convergence properties of the gradient descent algorithm.

B.2.2 Levenberg-Marquardt algorithm

The Levenberg-Marquardt algorithm is a combination of both previously introduced algorithms, the gradient descent and Gauss-Newton. It has been introduced in two steps, the first by Levenberg which is to combine both the gradient descent with the Gauss-Newton and the second by Marquardt which is to optimize the convergence rate by introducing a more dynamic damping.
The equation ***(B.19)*** states the basic Levenberg algorithm, which introduces the damping variable λ to the standard Newton-Gauss algorithm of equation ***(B.18)***.

$$\underline{x}^{(n+1)} = \underline{x}^{(n)} - \left(J_r^T J_r + \lambda I\right)^{-1} \cdot J_r r(\underline{x}) \tag{B.19}$$

The I in equation ***(B.19)*** denotes the identity matrix. One can easily see, that for big λ's the term $J_r^T J_r$ in $\left(J_r^T J_r + \gamma I\right)^{-1}$ can be disregarded. So if one considers $\gamma := (\lambda I)^{-1}$ we obtain the same equation like ***(B.6)*** for the gradient descent algorithm. On the other hand, for a small λ we obtain $(J_r^T J_r)^{-1}$ for the significant term which leads to the Gauss-Newton algorithm as given in ***(B.18)***.
If one contemplates the Levenberg algorithm from the ordinary Gauss-Newton algorithm, a big λ factor results into shorter steps each iteration. Therefore the Levenberg algorithm can be seen as some kind of damped Gauss-Newton algorithm. The goal is now to choose the λ in a way that equation ***(B.19)*** comes closer to the gradient descent algorithm as long the position $\underline{x}^{(n)}$ is far away from a minimum, because the gradient descent algorithm provides better convergence properties. With a $\underline{x}^{(n)}$ close to the local minimum the λ factor should shrink. Because under this circumstances one profits from the fast convergence properties of the Gauss-Newton algorithm, which has approximately quadratic convergence rate and no "ping pong"-effect.
The simplest strategy to set λ is to introduce a constant factor ν. Now we decrease the λ-factor in each step of iteration by dividing by ν. Additionally we setup the constraint that for each step of iteration the next function value of $\underline{x}^{(n+1)}$ never exceed the value of $\underline{x}^{(n)}$, equation ***(B.20)***.

$$f(\underline{x}^{(n+1)}) < f(\underline{x}^{(n)}) \tag{B.20}$$

If the constraining inequality ***(B.20)*** is not true for a step, λ will be iteratively increased by multiplying with ν until the constraint ***(B.20)*** is true again. A second more comprehensive approach is to use Trust-Region methods to determine the damping. For the Trust-Region we take a deeper look at an other characteristic of the approximation, called the gain ratio. The gain ratio is the proportion between the model and the real functions behavior given as in equation ***(B.21)***.

$$\varrho_n = \frac{f(\underline{x}^{(n)}) - f(\underline{x}^{(n+1)})}{G(\underline{x}^{(n)}, \underline{x}^{(n+1)})} \tag{B.21}$$

$with$

$$G(\underline{x}) = \frac{1}{\mu} \left(J^T f(\underline{x})\right)^T \left(J^T f(\underline{x})\right) \tag{B.22}$$

The rest of this method depends on two thresholds and a constant factor ν. The factor ν is usually set in between 2 and 5. If the gain ration falls underneath 0, what usually indicates a bad approximation, it is invalid. Further detail regarding the derivation can be found in [76]. Therefore the iteration will be stopped at the current position $\underline{x}^{(n)}$ and λ will be increased, by multiplying it with ν, until the gain ration raises above 0 again. On the other hand for a $\varrho > 0$ the iteration continues normally. Within each step of iteration the λ-factor then will be decreased by multiplying it with $max\left\{\frac{1}{3}, 1-(2\varrho-1)^3\right\}$.
Marquardt concerned, that with the fading form Gauss-Newton to gradient descent by a rising λ factor, the Hessian approximation $J_r^T J_r$ looses influence. Therefore he suggested not to use the identity matrix for multiplication with λ but with the diagonal matrix $diag(J_r^T J_r)$. Therefore we can rewrite equation ***(B.19)*** to the Levenberg Marquardt algorithm given as ***(B.23)***.

$$\underline{x}^{(n+1)} = \underline{x}^{(n)} - \left(J_r^T J_r + \lambda diag(J_r^T J_r)\right)^{-1} \cdot J_r r(\underline{x}) \tag{B.23}$$

With the given formula ***(B.23)*** one now has a stable and efficient method to solve problems like given with the arbitrary path correction of equation ***(4.28)*** in section *[B]*.

Appendix C

Transformations

Transformations can be expressed in multiples ways. The most common and also human understandable is the representation as a vector. Such vectors contain 6 components, where usually the first 3 describe the translation between the actual and the described frame and the final 3 components the rotations. This representation is very compact, but the major problem is that this description is not unique. There exist multiple ways to interpret the rotation information within this vector. These interpretations can be categorized into two groups, the so called Euler angles and Tait-Bryan angles which themselves provide multiple different notations:

- **Proper Euler angles** (z-x-z, x-y-x, y-z-y, z-y-z, x-z-x, y-x-y)
- **Tait-Bryan angle** (x-y-z, y-z-x, z-x-y, x-z-y, z-y-x, y-x-z)

Hence to be able to interprete the rotation correctly it is essential to know which notation to use, because each will end up in something completely different[1] than the other.
Yet, there are other ways to describe transformations and especially rotations in a unique way. The most common way for a unique description are matrices. One of these matrix based notation is the so called homogeneous transformation matrix. The homogeneous transformation matrix is combining both, translation and rotations in a 4 dimensional (4×4) homogeneous matrix usually denoted with $\mathbf{T}$. Since both parts, the translation as well as the rotation are packed into a single matrix $\mathbf{T}$, it is quite simple to transform a simple point $\underline{p}$ by just multiplying it with $\mathbf{T}$:

$$\mathbf{T} \cdot \underline{p} = \hat{\underline{p}} \tag{C.1}$$

Sometimes rotations and translations are handled in separate matrices, where the rotation then is represented by a 3x3 matrix usually denoted with $\mathbf{R}$ and a 3×1 vector $\underline{t}$ containing the translation. In order to represent the same transformation, like in equation ***(C.1)*** above, one needs to perform the following two step calculation:

$$\mathbf{R} \cdot \underline{p} + \underline{t} = \hat{\underline{p}} \tag{C.2}$$

[1] For small angles the values actually tend to look quite similar, but as soon as the angles grow larger the differences between the notations also rise.

Sometimes transformations are also coded within Dual-Quaternions. Quaternions provide some benefit especially when it comes to rotations. A quaternion is a vector within a 4 dimensional hyperbolic space, with 3 complex dimensions. The subset of all unit quaternions then form a 4 dimensional sphere within this space. The basic idea founded on Euler's rotation theorem, which states:

> When a sphere is moved around its center it is always possible to find a diameter whose direction in the displaced position is the same as in the initial position.

One can imagine a quaternion as a vector pointing into the direction of rotation where the vector itself can be rotated around the direction where it is pointing. This finally results into

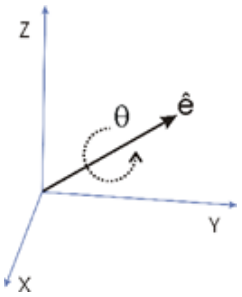

Figure C.1: Representation of an quaternion as vector and rotation

three components, giving the unit vectors direction and one angle Θ describing the rotation around this vector. To retrace this idea one can image a sphere e.g. a ball, where a (large) needle is stuck inside. The peek of the needle needs to be imagined sitting right at the center of the ball and it's tail is visible on the outer hull. If the ball will now be rotated around this center the peek keeps it position, but the tail of the needle moves around and its position can be described by the first three components. Additional on can also try to keep the position of the needles tail in room space and he still has one degree of freedom left for moving the ball. This will be turning around the axis the needle is pointing. After all, there will be no other way to rotate the ball around the origin (within 3 dimension space), in order to perform a different motion of the needle, as the two described above. But still, a quaternion is not just 4 for dimensional vector. They form a special algebra, which provides special properties related to interpolation and transformation of rotations which makes it quite simple to interpolate from one orientation to an other. Finally, the vector notations are only serving visualization purpose, since they usually are interpretable by humans. If somebody gets told that there is a 50° rotation around Z and 30° around X, he will have some idea about the constellation. But on the other hand if there is just a transformation matrix or a quaternion, it might be hard understand immediately what rotations are encoded, unless they don't represent plain 90° rotations. Nevertheless the matrix and quaternion representation proofed to be quite efficient within direct calculations on the machines, so both representations are necessary.

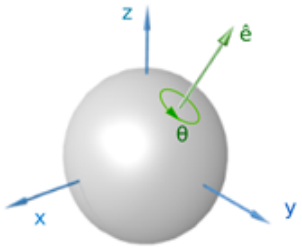

Figure C.2: Representation of an quaternion as vector and rotation

C.1 Deriving a 2 dimensional rotation matrix

In order to derive the equations for 2 dimensional rotation matrix we begin with a very simple, straight forward case, where a point $\underline{x}$ lies directly on the X-Axis so its components can be considered as $(x_0, 0)^T$. Any rotated position of this point around the origin will be on the radius x_0. Therefore the point will move on a circular path if rotated from $0°$ to $360°$ around the origin. Further, for any angle α there exists one right triangle, where the hypotenuse is

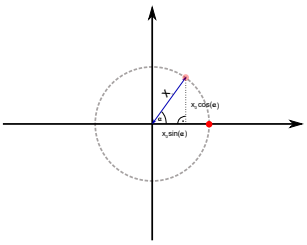

Figure C.3: Representation of an quaternion as vector and rotation

connecting the rotated point with the origin, and the catheti are aligned to the coordinate frames axis ***(Fig. C.3)***. Hence by knowing the desired angle α it will be easy to determine the rotated points coordinates by just applying basic trigonometric rules.

$$\hat{x} = x_0 cos(\alpha) \tag{C.3}$$

$$\hat{y} = x_0 sin(\alpha) \tag{C.4}$$

For this simple case where the original point is positioned directly on the X-Axis equations ***(C.3)*** and ***(C.4)*** provide a fairly simple scheme to rotate the given point around the origin. But what if the given point to rotate has not such a nice position? In order to rotate an arbitrary point, we consider a setup like illustrated in figure ***(Fig. C.4)***. Clearly, if one considers the triangle spanned between the tree points $\underline{\hat{p}}$, $\underline{p}$ and the origin, neither any axis is aligned to the coordinate frames axis, nor is it a right triangle. Hence the formulas presented

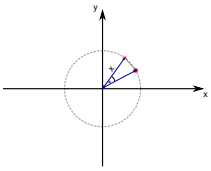

Figure C.4: Rotation of an arbitrary point

in equation ***(C.3)*** and ***(C.4)*** won't apply in the given scenario. In order to find a solution for the given problem, we will introduce the angle β, which is the angle between the new point and the x-axis ***(Fig. C.5)***. Furthermore it is clear, that β must be the combination of α and

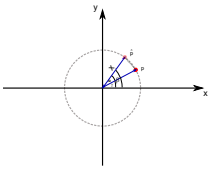

Figure C.5: Introducing the angle β and γ

γ, which is $\beta = \alpha + \gamma$. Assuming that all angles are well known, one can directly derive $\underline{\hat{p}}$ and $\underline{p}$ by applying equation ***(C.3)*** and ***(C.4)***.

$$\underline{\hat{p}} = \begin{pmatrix} x_0 cos(\beta) \\ x_0 sin(\beta) \end{pmatrix} = \begin{pmatrix} \hat{p_x} \\ \hat{p_y} \end{pmatrix} \tag{C.5}$$

$$\underline{p} = \begin{pmatrix} x_0 cos(\gamma) \\ x_0 sin(\gamma) \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \end{pmatrix} \tag{C.6}$$

By consulting any mathematical formulary like, one can find the following laws ***(C.7)*** and ***(C.8)*** concerning the addition of arguments within the sine and cosine function.

$$\sin(\alpha \pm \beta) = \sin(\alpha)\cos(\beta) \pm \cos(\alpha)\sin(\beta) \tag{C.7}$$

$$\cos(\alpha \pm \beta) = \cos(\alpha)\cos(\beta) \mp \sin(\alpha)\sin(\beta) \tag{C.8}$$

And since in $\beta = \alpha + \gamma$, the argument β can be removed from ***(C.5)*** by $\alpha + \gamma$ and the above rules ***(C.7)*** and ***(C.8)*** can be applied. ***(C.4)***.

$$\begin{aligned}
\underline{\hat{p}} &= \begin{pmatrix} x_0 cos(\beta) \\ x_0 sin(\beta) \end{pmatrix} \\
&= \begin{pmatrix} x_0 cos(\alpha+\gamma) \\ x_0 sin(\alpha+\gamma) \end{pmatrix} \\
&= \begin{pmatrix} x_0 \left(\cos(\alpha)\cos(\gamma) - \sin(\alpha)\sin(\gamma)\right) \\ x_0 \left(\sin(\alpha)\cos(\gamma) + \cos(\alpha)\sin(\gamma)\right) \end{pmatrix} \\
&= \begin{pmatrix} \cos(\alpha)\left[x_0\cos(\gamma)\right] - \sin(\alpha)\left[x_0\sin(\gamma)\right] \\ \sin(\alpha)\left[x_0\cos(\gamma)\right] + \cos(\alpha)\left[x_0\sin(\gamma)\right] \end{pmatrix} \\
&= \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} \begin{pmatrix} x_0\cos(\gamma) \\ x_0\sin(\gamma) \end{pmatrix}
\end{aligned} \tag{C.9}$$

In equation ***(C.9)*** the formula had already been separated into a matrix-vector product. Finally by taking a closer look to the right hand vector of equation ***(C.9)*** and recalling the result of equation ***(C.6)*** it is very simple to derive the final result ***(C.11)*** for an rotation within the x/y - plane.

$$\underline{\hat{p}} = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} \begin{pmatrix} p_x \\ p_y \end{pmatrix} \tag{C.10}$$

$$= \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} \underline{p} \tag{C.11}$$

With equation ***(C.11)*** any arbitrary point in the 2 dimensional X/Y-plane can be rotated around the origin. The next logical step would be to extend this to the 3 dimensional space. Figure ***(Fig. C.6)*** depicts the perpendicular planes within the 3 dimensional euclidean space with a right handed coordinate frame. As one can see, the Z/Y-plane is exactly equal to the X/Y-plane and hence the equations in ***(C.11)*** will apply 1:1 to it. Finally there is the X/Z-plane, which appears to be mirrored about the vertical axis compared to the two others. Hence the equations given in ***(C.11)*** need to be slightly adjusted.
First thing to be recognized is that the positive rotation within the X/Z-plane is now inverted to the other planes. Figure ***(Fig. C.7)*** illustrates this by aligning the X and Z axis in the same way as the axis for the X/Y and Z/Y planes in figure ***(Fig. C.6)***. The perpendicular Y direction is now pointing into the opposite direction and therefore the positive rotation turns around. Hence for a point on the X-axis x_0 the positive rotation about α will be going into negative Z direction ***(C.12)*** & ***(C.13)***.

$$\hat{x} = x_0 cos(\alpha) \tag{C.12}$$

$$\hat{z} = -x_0 sin(\alpha) \tag{C.13}$$

Hence the equations for an arbitrary point for the X/Z-plane, with the same setup of the angles with $\beta = \alpha + \gamma$, is given by equation ***(C.14)***.

$$\begin{aligned}
\hat{\underline{p}} &= \begin{pmatrix} x_0 cos(\beta) \\ -x_0 sin(\beta) \end{pmatrix} \\
&= \begin{pmatrix} x_0 cos(\alpha+\gamma) \\ -x_0 sin(\alpha+\gamma) \end{pmatrix} \\
&= \begin{pmatrix} x_0 \left(\cos(\alpha)\cos(\gamma) - \sin(\alpha)\sin(\gamma)\right) \\ -x_0 \left(\sin(\alpha)\cos(\gamma) + \cos(\alpha)\sin(\gamma)\right) \end{pmatrix} \\
&= \begin{pmatrix} \cos(\alpha)\left[x_0\cos(\gamma)\right] + \sin(\alpha)\left[-x_0\sin(\gamma)\right] \\ -\sin(\alpha)\left[x_0\cos(\gamma)\right] + \cos(\alpha)\left[-x_0\sin(\gamma)\right] \end{pmatrix} \\
&= \begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix} \begin{pmatrix} x_0\cos(\gamma) \\ -x_0\sin(\gamma) \end{pmatrix} \\
&= \begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix} \begin{pmatrix} p_x \\ p_z \end{pmatrix}
\end{aligned} \tag{C.14}$$

Finally, the rotations in all planes in 3 dimensional space are covered and the equations can be rewritten for a general 3 dimensional point $\underline{p} = (p_x, p_y, p_z)^T$.

$$\hat{\underline{p}} = \mathbf{R}_z(\gamma) \cdot \underline{p} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma) \\ 0 & \sin(\gamma) & \cos(\gamma) \end{pmatrix} \cdot \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \tag{C.15}$$

$$\hat{\underline{p}} = \mathbf{R}_y(\beta) \cdot \underline{p} = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{pmatrix} \cdot \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \tag{C.16}$$

$$\hat{\underline{p}} = \mathbf{R}_x(\alpha) \cdot \underline{p} = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \tag{C.17}$$

Since equation ***(C.15)***-***(C.17)*** are giving the separate rotations, one can achieve a complete function about all angles by simply multiplying them $\mathbf{R}(\alpha, \beta, \gamma) = \mathbf{R}_z(\gamma) \cdot \mathbf{R}_y(\beta) \cdot \mathbf{R}_x(\alpha)$ which would be the Tait-Bryan zyx notation. This notation is often also called RPY (Roll Pitch Yaw) and is the de facto standard in automotive robotics.

$$\mathbf{R}(\alpha, \beta, \gamma) = \begin{bmatrix} c(\gamma)c(\beta) & c(\gamma)s(\beta)s(\alpha) - c(\alpha)s(\gamma) & s(\gamma)s(\alpha) + c(\gamma)c(\alpha)s(\beta) \\ c(\beta)s(\gamma) & c(\gamma)c(\alpha) + s(\gamma)s(\beta)s(\alpha) & c(\alpha)s(\gamma)s(\beta) - c(\gamma)s(\alpha) \\ -s(\beta) & c(\beta)s(\alpha) & c(\beta)c(\alpha) \end{bmatrix} \tag{C.18}$$

C.2 Affine transformations in homogeneous coordinates

An affine transformation in $\mathbb{R}^n$ euclidean space is a transformation which

- **preserves lines**.
  If a set of points forms a line they also do in the transformed set.

- **preserves parallelism**.
  If there are parallel lines transformed they will remain parallel.
- **preserves the ratio**.
  If there are multiple points on a line, the ratio of the distances between points will be preserved after transformation.

Considering now the above described rotations they clearly satisfy the fundamental requirements of an affine transformation. In general one can find affine transformations in the euclidean $\mathbb{R}^n$ space, that not only rotates but also shear, mirror and scale points. Furthermore they can all be presented by a linear transformation within $\mathbb{R}^n$. Unfortunately translating the points, which also is a affine transformation, is not possible by a linear transformation within $\mathbb{R}^n$. Hence an affine transformation in general is allays described as given in equation ***(C.19)***.

$$\mathbf{A} \cdot \underline{p} + \underline{t} = \underline{\hat{p}} \tag{C.19}$$

In equation ***(C.19)*** $\underline{p}$ is a point in $\mathbb{R}^n$, $\mathbf{A}$ is a $\mathbb{R}^{n \times n}$ matrix representing the linear transformation and $\underline{t}$, also $\mathbb{R}^n$, is the translation vector. However, in many situations it will be more practically to just have one matrix, say $\mathbf{T}$, which contains the whole transformation. An example of such a situation is where multiple transformations will be chained to a single transformation, like equation ***(C.20)***, or when one just needs to get the inverse of a certain transformation, which just turns out to be $\mathbf{T}^{-1}$.

$$\begin{aligned} \underline{\hat{p}} &= \mathbf{T}_1 \cdot \mathbf{T}_2 \cdot ... \mathbf{T}_n \cdot \underline{p} \\ &= \mathbf{T}_x \cdot \underline{p} \end{aligned} \tag{C.20}$$

where

$$\mathbf{T}_x = \mathbf{T}_1 \cdot \mathbf{T}_2 \cdot ... \mathbf{T}_n$$

Since $\mathbf{T}$ can not be within $\mathbb{R}^{n \times n}$, as discussed above it needs to be extended by one dimension and thus $\mathbf{T}$ needs to be within $\mathbb{R}^{(n+1)\times(n+1)}$. The transformation matrix as given in equation ***(C.21)***.

$$\mathbf{T} = \begin{pmatrix} \mathbf{R} & \underline{t} \\ 0 & 1 \end{pmatrix} \tag{C.21}$$

As one can see in equation ***(C.21)*** the last row of the transformation matrix will be the vector $(0 \cdots 0 \quad 1)$. Hence the vector $\underline{p}$ needs also be extended for being 4×1. This turns $\underline{p}$ technically speaking into a so called **homogeneous coordinate**. A homogeneous coordinate is an point in an projective space. But as long one keeps the final row of the transformation as $(0 \cdots 0 \quad 1)$ and extends the point vector by 1, such a transformation ends up to be a simple affine transformation within $\mathbb{R}^n$. The benefit though is that this affine transformation is now wrapped up by a simple multiplication of the form $\mathbf{T} \cdot \underline{p}$.
By changing the 1 either in the transformation at position $(n+1, n+1)$, or in the last element of the point, one achieves something like a scaling effect, since the hyperplane in which the coordinates are embedded is shifted within the projective space. However, as long as all

points remain within a single hyperplane within the projective space this transformations always appear to be affine transformations. Only by changing the null-vector in the last row of the transformation the points will be spread over multiple hyperplanes within the projective space, which turns the resulting transformation into a non affine transformation with distortion effects.

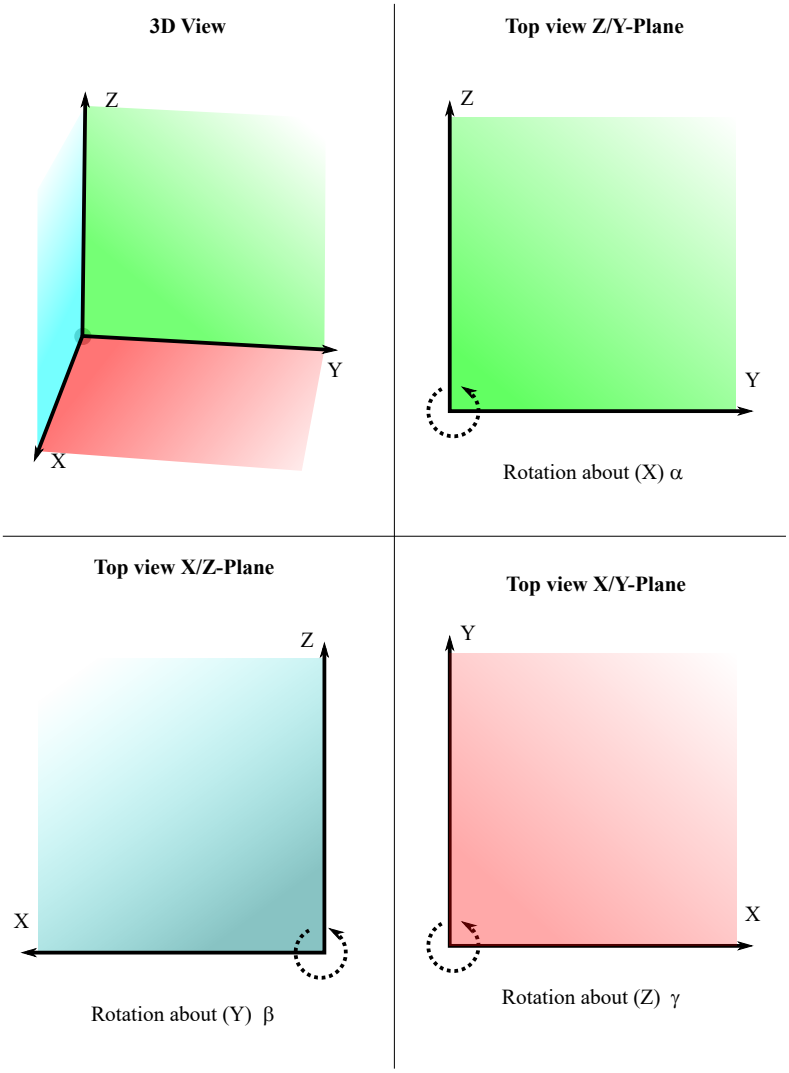

Figure C.6: All perpendicular planes in 3 dimensional space

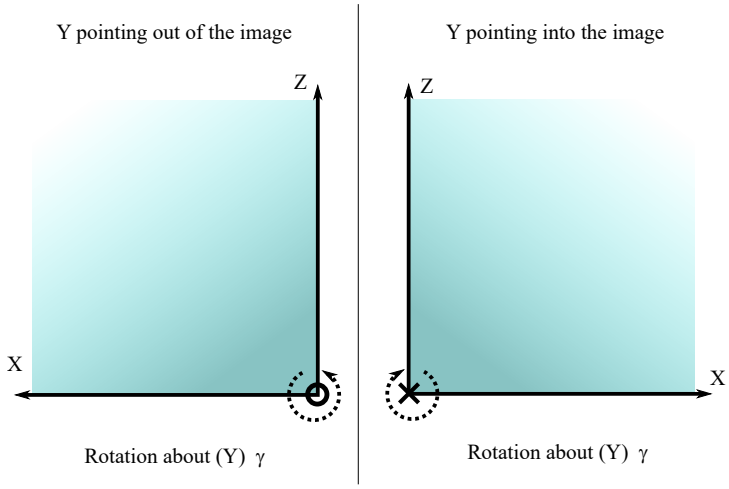

Figure C.7: Mirroring the X/Z-plane

Bibliography

[1] F. W. Taylor. *The Principles of Scientific Management,*. Harper & Row, Publishers, Incorporated, 1911.

[2] S. K .Saha. *Introduction to Robotics, 2e*. Tata McGraw-Hill Education.

[3] J. Denavit and R. S. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *Trans. of the ASME. Journal of Applied Mechanics*, 1955.

[4] M. Herrmann, C. Dahlke, M. Otesteanu, and Marc-Andre Otto. An over-determined path correction algorithm for sparse dimensional measurements. *Recent Advances in Circuits, Communications and Signal Processing*, 2013.

[5] N. Pears, Y. Liu, and P. Bunting. *3D Imaging, Analysis and Applications*. Springer London, 2012.

[6] A. E. Conrady. Decentred lens-systems. *Monthly notices of the Royal Astronomical Society*, 1919.

[7] D. C. Brown. Decentering distortion of lenses. *Photogrammetric Engineering*, May 1966.

[8] G. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics Series B Numerical Analysis*, 1965.

[9] Y. I. Abdel-Aziz, H. M. Karara, and M. Hauck. Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry. *Proceedings of the Symposium on Close-Range photogrammetry*, 1971.

[10] T. Marzan, Genaro, and H.M. Karara. A computer program for direct linear transformation solution of the colinearity condition, and some applications of it. *[No source information available]*, 1975.

[11] R. Y. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal on Robotics and Automation*, 1987.

[12] F. Zheng and B. Kong. Calibration of linear structured light system by planar checkerboard. *International Conference on Information Acquisition, 2004. Proceedings.*, 2004.

[13] I. Alan. Laser stripe profiler calibration using nonlinear models. 2001.

[14] J. Santolaria, J. Pastor, F.J. Brosed, and J. Aguilar. A one-step intrinsic and extrinsic calibration method for laser line scanner operation in coordinate measuring machines. *Measurement Science and Technology*, 2009.

[15] P. Mansbach. Calibration of a camera and light source by fitting to a physical model. *Computer Vision, Graphics, and Image Processing*, 1986.

[16] H. Zhuang, Z. S. Roth, and R. Sudhakar. Simultaneous robot/world and tool/flange calibration by solving homogeneous transformation equations of the form ax=yb. *IEEE Transactions on Robotics and Automation*, 1994.

[17] S.G. Aristos and D. Tzafestas. An extensive analysis of the simultaneous world and hand-eye calibration problem. *International Journal of Factory Automation,Robotics and Soft Computing*, 2008.

[18] Z. Gan and Q. Tang. *Visual Sensing and its Applications: Integration of Laser Sensors to Industrial Robots*. Springer Berlin Heidelberg, 2011.

[19] P. Roebrock, K. Boehnke, and P.Rives. Offline path correction system for industrial robots. *9th WSEAS International Conference on Automatic Control,Istanbul*, 2007.

[20] SRI International. Artificial Intelligence Center, Agin, and G.J. *Real time control of a robot with a mobile camera*. Technical note. SRI International, 1979.

[21] T. Shu, S. Gharaaty, W. Xie, A. Joubair, and I. A. Bonev. Dynamic path tracking of industrial robots with high accuracy by visual servoing. *2017 12th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 2017.

[22] F. Castelli, S. Michieletto, S. Ghidoni, and E. Pagello. A machine learning-based visual servoing approach for fast robot control in industrial setting. *International Journal of Advanced Robotic Systems*, 2017.

[23] J. A. Piepmeier, G. V. McMurray, and H. Lipkin. Uncalibrated dynamic visual servoing. *IEEE Transactions on Robotics and Automation*, 2004.

[24] A. Astolfi, L. Hsu, M. S. Netto, and R. Ortega. Two solutions to the adaptive visual servoing problem. *IEEE Transactions on Robotics and Automation*, 2002.

[25] J. Qu, F. Zhang, H. Xu andG. Li, and S. Guo. Adaptive neural network visual servoing of dual-arm robot for cyclic motion. *Industrial Robot: An International Journal*, 2017.

[26] F. Castelli, S. Michieletto, S. Ghidoni, and E.Pagello. A machine learning-based visual servoingapproach for fast robot control inindustrial setting. *International Journal of Advanced Robotic Systems*, 2017.

[27] S. Ching-Long and Y. Lee. A simple robotic eye-in-hand camera positioning and alignment control method based on parallelogram features. *Robotics 2018*, 2018.

[28] F. Chaumette and F. Hutchinson. Visual servo control, part i: Basic approaches. *IEEE Robotics and Automation Magazine*, 2006.

[29] H. Michel and P. Rives. Singularities in the determination of the situation of a robot effector from the perspective view of 3 points. Research report, 1993.

[30] D. Leon, Emmanuel, and C. Gordon. A new method for solving 6d image-based visual servoing with virtual composite camera model. 2015.

[31] Al-Junaid and Hessa. Ann based robotic arm visual servoing nonlinear system. *Procedia Computer Science*, 62:23–30, 12 2015.

[32] D. Holz, A. Topalidou-Kyniazopoulou, J. StÃ¼ckler, and S. Behnke. Real-time object detection, localization and verification for fast robotic depalletizing. *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1459–1466, 2015.

[33] C. Pfitzner, W. Antal, P. Hess, S. May, C. Merkl, P. Koch, R. Koch, and M. Wagner. 3d multi-sensor data fusion for object localization in industrial applications. *ISR/Robotik 2014; 41st International Symposium on Robotics*, 2014.

[34] R.A. Jarvis. A perspective on range finding techniques for computer vision. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 1983.

[35] H. Yang, X. Liu, and I. Patras. A simple and effective extrinsic calibration method of a camera and a single line scanning lidar. *Pattern Recognition (ICPR), 2012 21st International Conference on*, 2012.

[36] A. Mavrinac, X. Chen, P. Denzinger, and M. Sirizzotti. Calibration of dual laser-based range cameras for reduced occlusion in 3D imaging. *IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, 41:79–83, 2010.

[37] I. Reid. Projective calibration of a laser-stripe range finder. *Image and Vision Computing*, 1996.

[38] J. Davis and X. Chen. A laser range scanner designed for minimum calibration complexity. *In Third International Conference on 3D Digital Imaging and Modeling*, pages 91–98, 2001.

[39] J. L. Vilaca, J. Francisco Cruz Fonseca, and A. C. M. Pinho. Calibration procedure for 3D measurement systems using two cameras and a laser line. *Optics & Laser Technology*, 2009.

[40] D. K. Naidu. A comparative analysis of algorithms for determining the peak position of a stripe to subpixel accuracy. *Proc. British Machine Vision Conf*, 1991.

[41] W. Yibo, H. Zhongxiang, Y. Yao, Y. Junwei, and T. Jiaxu. A Method for Laser-line Detection in a Three-dimensional Measurement System. *ISTM/2009: 8TH INTERNATIONAL SYMPOSIUM ON TEST AND MEASUREMENT, VOLS 1-6*, 2009.

[42] W. Qing-yang, S. Xian-yu, L. Jing-zhen, and H. Bin. A New Method for Extracting the Centre-line of Line Structure Light-stripe. *Journal of Sichuan University(Engineering Science Edition)*, 2007.

[43] R.W. Shonkwiler. *Finance with Monte Carlo*. Springer Undergraduate Texts in Mathematics and Technology. Springer New York, 2013.

[44] I. Karatzas and S. E. Shreve. *Brownian Motion and Stochastic Calculus*. Springer New York, 1991.

[45] B.K. Øksendal. *Stochastic Differential Equations: An Introduction with Applications*. Springer, 2010.

[46] W.E. Snyder and H. Qi. *Machine Vision*. Cambridge University Press, 2004.

[47] S. Ambwani and State University of New York at Buffalo. *Map Based Stochastic Methods for Joint Estimation of Unknown Image Degradation Parameters and Super-resolution*. State University of New York at Buffalo, 2006.

[48] M. Herrmann, M. Otesteanu, and M. Otto. Fast and robust point cloud matching based on em-icp prepositioning. *2012 10th International Symposium on Electronics and Telecommunications*, 2012.

[49] P. J. Besl and H. D. McKay. A method for registration of 3-D shapes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 1992.

[50] S. Gold, A. Rangarajan, C. P. Lu, S. Pappu, and E. Mjolsness. New algorithms for 2D and 3D point matchingpose estimation and correspondence. *Pattern Recognition*, 1998.

[51] S. Granger and X. Pennec. Multi-scale em-icp: A fast and robust approach for surface registration. *European Conference on Computer Vision (ECCV 2002), volume 2353 of LNCS*, 2002.

[52] T. Tamaki, M. Abe, B. Raytchev, and K. Kaneda. Softassign and em-icp on gpu. *Proceedings of the 2010 First International Conference on Networking and Computing*.

[53] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1972.

[54] M. A. Greenspan and M. Yurick. Approximate k-d tree search for efficient icp. 2003.

[55] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 1973.

[56] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 1977.

[57] S. Granger, X. Pennec, and A. Roche. Rigid point-surface registration using an em variant of icp for computer guided oral implantology. *Proceedings of the 4th International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2001.

[58] J. L. Bentley. Multidimensional binary search trees used for associative searching. 1975.

[59] I. Wald and V. Havran. On building fast kd-trees for ray tracing, and on doing that in o(n log n). *IN PROCEEDINGS OF THE 2006 IEEE SYMPOSIUM ON INTERACTIVE RAY TRACING*, 2006.

[60] B. K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 1987.

[61] B.Espiau, F.Francois, and P.Rives. A new apporach to visual servoing in robotics. 1994.

[62] P. I. Corke. High-performance visual closed-loop robot control,. *Department of Mechanical and Manufacturing Engineering*, 1994.

[63] M. Bonkovi, A. Hace, and K. Jezernik. A new method for uncalibrated visual servoing. *9th IEEE International Workshop on Advanced Motion Control, 2006.*, 2006.

[64] B. K. P. Horn. Tsais camera calibration method revisited. *MIT Press, Cambridge, Massachusetts and McGraw-Hill, New York.*

[65] A and Ranganathan. The Levenberg-Marquardt algorithm, 2004.

[66] H. Stark and J. W. Woods. *Probability, Random Processes, and Estimation Theory for Engineers*. Prentice-Hall, Inc., 1986.

[67] R.Y. Tsai. An efficient and accurate camera calibration technique for 3D machine vision. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Miami Beach, FL*, 1986.

[68] K. S. Arun, T. S. Huang, and s. D. Blostein. Least-Squares Fitting of Two 3-D Point Sets. *IEEE Trans. Pattern Anal. Mach. Intell.*, 1987.

[69] P. Roebrock. *Multi-sensor controlled assembly and application with manipulators.* Universitatii Politehnica din Timisoara Seria Electronica si Telecomunicatii, 2009.

[70] G.J. Agin. *The solution of nonlinear inverse problems and the Levenberg-Marquardt method*. 2007.

[71] M. Herrmann and M. Otesteanu. A map estimator based on geometric brownian motion for sample distances of laser triangulation data. *Optics and Lasers in Engineering*, 2016.

[72] H. Herrmann and A. Krzizok. *Photogrammetrie, Laserscanning, optische 3D-Messtechnik : Beiträge der Oldenburger 3D-Tage 2010*. Berlin : Wichmann.

[73] M. Herrmann, M. Otesteanu, and M. Otto. A method to determine the extrinsic parameter of laser triangulation sensors, with restricted mobility. *2014 11th International Symposium on Electronics and Telecommunications (ISETC)*, 2014.

[74] M. Herrmann, M. Otesteanu, and M. Otto. A novel approach for automated car body panel fitting. *2013 IEEE International Symposium on Robotic and Sensors Environments (ROSE)*, 2013.

[75] K. Vijay, A. K. RohatgiBerthold, and Md. Ehsanes Saleh. An introduction to probability and statistics. *Wiley Series in Probability and Statistics*.

[76] K. Madsen, H. B. Nielsen, and O. Tingleff. Methods for non-linear least squares problems (2nd ed.). 2004.