Sensor based real-time control of robots

Andersen, Thomas Timm; Ravn, Ole; Andersen, Nils Axel

Technical University of Denmark

Thomas Timm Andersen

Sensor based real-time control of robots

PhD Thesis, November 2015

DTU Electrical Engineering
Department of Electrical Engineering

Sensor based real-time control of robots

Thomas Timm Andersen

Technical University of Denmark
Kgs. Lyngby, Denmark, 2015

Technical University of Denmark
Automation and Control (AUT)
Richard Petersens Plads 326
DK-2800, Kgs. Lyngby
Denmark
Phone: (+45) 45 25 35 76
Email: info@elektro.dtu.dk
www.elektro.dtu.dk

Summary

As robots are becoming more and more widespread in manufacturing, the desire and need for more advanced robotic solutions are increasingly expressed. This is especially the case in Denmark where products with natural variances like agricultural products takes up a large share of the produced goods. For such production lines, it is often not possible to use primitive preprogrammed industrial robots to handle the otherwise repetitive tasks due to the uniqueness of each product.

To handle such products it is necessary to use sensors to determine the size, shape, and position of the product before a proper trajectory can be calculated in real-time for the robot to execute. This introduces a multitude of different challenges, some of which this project seeks to find the answer to.

The production environment of agricultural products is not very well suited for advanced machinery. Handling crops often releases a lot of dust, livestock releases bodily fluids, and all naturally grown products plays host to different kinds of bacterial flora. To ensure food safety it is thus necessary to clean the production facilities daily. This is often done with high-pressure water which can easily cause small changes in the position or orientation of sensors and robots if hit, which in turn corrupts their internal relative calibration. And if the entire robot motion is based on a miscalibrated sensor measurement, the end result could easily be suboptimal or destroyed products, or even destroyed machinery.

To avoid such outcomes and thus make sensor based control more reliable, an accurate calibration method has been developed as part of this project. After initial placement of a calibration target mounted on the robot end effector under a laser range scanner, the method can autonomously control the robot to determine the transformation between the laser scanner and the robot. And once the robot has a rough idea of the position of the scanner, the method can be used complete autonomously to correct for small misalignment after the daily cleaning cycle.

Furthermore, the method makes it possible to calculate the worst case error of the calibration. This can help in guaranteeing end product uniformity, i.e. as part of a ISO9000 certification.

Once the robot knows the pose of the product that needs manipulation, it needs to do a real-time calculation of an appropriate trajectory. The trajectory does not only need to be accurate with respect to the end pose of the robot, it also needs to be temporally accurate so the robot can manipulate the product without stopping the conveyor belt and thus possibly the entire production.

To achieve temporal accuracy, it is necessary to know the delay throughout the entire system from acquisition delays in the sensor to actuation delays in the robot. To that end a method for measuring the actuation and response delay of an industrial robot manipulator, relative to the joint configuration of the robot, is presented. It is also shown how modern machine learning algorithms can be trained to build model based on the measurements.

Once a model of the delay is constructed, it is furthermore shown how the model can be used for both forward and inverse predictions as well as current state corrections and thus improve on the temporal accuracy of an industrial robot manipulator.

When using predefined trajectories for the robot, it is possible to simulate every motion and through prediction minimize the number of issues to ensure high uptime. With real-time generated trajectories and varying product shapes, this is not possible to the same extend. The robot could end up in singular

configurations, the risk of a grasp failing when a product is lifted is increased, a sensor could malfunction or foreign objects could end up on the conveyor.

A production system needs to be able to handle all these issues to ensure robustness and high uptime of the production facility. To accomplish this it is shown how an expert system can be used to monitor a robot executing a task and ensure that the system can either handle issues or at least degrade in the least obstructive way. This is ensured through rules that defines the boundaries for solving the given task, and how the system must react if the boundary is crossed.

Due to the generality of the methods presented in this project they constitutes a significant contribution towards using sensors for real-time control of robots, both in conjunction with industrial robots as well as in other robotic contexts.

Resumé

I takt med at udbredelsen af robotter i industrien øges, stiger ønsket og behovet for mere avancerede robotløsninger også. Dette er især tilfældet i Danmark, hvor produkter med en naturlig variation, såsom landbrugsprodukter, udgør en stor del af de producerede varer. Til sådan produktion er det ofte ikke muligt at benytte primitive forprogrammerede industrirobotter, da hvert enkelt produkt er forskelligt.

For at håndtere sådanne varierende produkter er det nødvendigt at bruge sensorer til at måle størrelse, form og position af hvert enkelt emne, således at en korrekt bane for robotten kan udregnes i realtid og derefter udføres. Dette introducerer en stribe forskellige udfordringer, hvortil dette projekt søger at løse nogle af dem.

Avancerede maskiner er generelt ikke særlig velegnede til det produktionsmiljø der ofte findes ved forarbejdning af landbrugsprodukter. Håndtering af afgrøder støver, dyrene afgiver kropsvæsker og der er en naturlig forekomst af mange forskellige bakterier på alle naturskabte produkter. Alt dette nødvendiggør en daglig rengøring af hele produktionsapparatet af hensyn til fødevaresikkerheden. Dette gøres ofte med højtryksspulere, hvilket kan medføre at sensorer og robotter drejer eller rykker sig en smule, hvis de bliver ramt. Denne forskydning ødelægger den relative kalibrering der er imellem de to apparater. Hvis robottens bevægelse dermed er baseret på en fejlkalibreret sensors målinger vil det medføre mindre udbytte eller deciderede ødelagte

slutprodukter, hvis det da ikke ligefrem resulterer i at produktionslinjen bliver beskadiget.

For at undgå sådanne hændelser, og dermed gøre sensorbaseret styring mere pålideligt, er der som en del af dette projekt udviklet en præcis kalibreringsmetode. Ved at montere en kalibreringsskive for enden af en robot og derefter bevæge den ind under en laserbaseret afstandsmåler, er kan metoden styre robotten autonomt og derved finde transformationen mellem robotten og afstandsmåleren. Og så længe at robotten har en omtrentligt ide om hvorhenne afstandsmåleren er, kan robotten fuldstændig autonomt foretaget en rekalibrering efter den daglige rengøring.

Derudover er det også muligt at beregne den største fejl der kan forekomme i kalibreringen. Derved kan en graden af ensformighed i slutproduktet lettere bestemmes, f.eks. i forbindelse med en ISO9000 certificering.

Når robotten først ved hvorhenne produktet den skal bearbejde befinder sig, skal den foretage en realtidsudregning af en passende bane. Denne bane skal ikke blot være præcis i forhold til position og orientering, den skal også være tidsmæssig præcis således at robotten kan bearbejde produktet uden at stoppe produktionslinjen og dermed potentielt hele produktionen.

For at sikre at timingen er korrekt er det nødvendigt at kende forsinkelserne igennem hele systemet, fra måleforsinkelser i sensorerne til aktiveringsforsinkelser i robotten. Til at gøre dette præsenteres en metode der kan måle aktiverings- og svarforsinkelserne for en industrirobot, afhængig af positionen af hvert enkelt af robottens led. Det er også vidst hvordan moderne maskinlæringsmetoder kan bruge de målte forsinkelser til at lære en model af forsinkelserne.

Det er også vist hvordan en sådan model kan bruges til direkte og invers forudsigelse af forsinkelser samt til at give en mere præcis beskrivelse af robottens nuværende tilstand, hvilket kan bruges til at forbedre timingen for industrirobotter.

Når man bruger forprogrammerede baner til at styre en robot, er det muligt at simulere hvert enkelt bevægelse. Derved kan man forudsige og efterfølgende minimere mængden af mulige problemer og således sikre en høj oppetid. Når man derimod bruger realtidsgenererede baner og skal håndtere produkter der varierer i form og størrelse er dette ikke muligt i samme omfang. Robotten kan komme ud i singulære positioner, risikoen for at robotten ikke får ordentlig fat og dermed mister sit greb i produktet er større, en sensor kan fejle eller et fremmedobjekt kan havne på transportbåndet.

Et produktionsapparat skal kunne håndtere alle disse problemer, således at det kører robust og derved sikrer en høj oppetid. For at opnå er det vist hvordan et ekspertsystem kan bruges til at overvåge en opgave og sikre at systemet enten kører videre uafhængigt at sådanne problemer, eller i hvert fald reagerer mest hensynsfuldt. Dette er opnået gennem regler der definerer grænserne for hvad der er nødvendigt for at løse en opgave og grænserne for hvordan en opgave kan løses, samt hvordan systemet skal reagere hvis en af disse regler ikke kan overholdes.

Metoderne der er præsenteret i dette projekt kan anvendes så bredt at de både i forbindelse med industrielle robotter og i andre sammenhænge bidrager væsentligt til at fremme sensorbaseret realtidsstyring af robotter.

Preface

This thesis is written as a partial fulfilment of the requirements for the PhD degree in engineering. The PhD project was conducted at the Technical University of Denmark's Department of Electrical Engineering in the Automation and Control group. The project was carried out from December 2012 to November 2015. The project was co-funded by the Innovation Fund Denmark through project "11-118482 Real-time controlled robots for the meat industry" and DTU.

The supervisors of the project were Associate Professor Ole Ravn (main supervisor) and Associate Professor Nils Axel Andersen, both from the Automation and Control group at DTU Electrical Engineering. Part of the research was conducted at the Cognitive Robotics Lab at Georgia Institute of Technology's College of Computing with Professor Henrik Iskov Christensen acting as supervisor.

The thesis constitutes a collection of papers, which has been submitted for conferences and journals during the project period as well as methods developed during further research.

Thomas Timm Andersen
Kongens Lyngby, November 2015

List of Publications

Papers included in the thesis

(A) T. T. Andersen, N. A. Andersen, and O. Ravn. “Exception detection and handling in mission control for mobile robots”. In: *Proceedings of 8th IFAC Symposium on Intelligent Autonomous Vehicles*. Vol. 8. IFAC Proceedings Volumes (IFAC-PapersOnline). Elsevier Science, 2013, pp. 187–192. *Published*

(B) T. T. Andersen, N. A. Andersen, and O. Ravn. “Calibration between a laser range scanner and an industrial robot manipulator”. In: *2014 IEEE Symposium on Computational Intelligence in Control and Automation (CICA)*. IEEE. 2014, pp. 1–8. *Published*

(C) T. T. Andersen, N. A. Andersen, and O. Ravn. “Optimizing the autonomous self-calibration between a robot and a distance sensor”. In: *Robotics and Computer-Integrated Manufacturing* (2016). *Submitted*

(D) T. T. Andersen, H. B. Amor, N. A. Andersen, and O. Ravn. “Measuring and Modelling Delays in Robot Manipulators for Temporally Precise Control using Machine Learning”. In: *14th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2015. *Presented*

Other publications

- H. Wu, W. Tizzano, T. T. Andersen, N. A. Andersen, and O. Ravn. “Hand-Eye Calibration and Inverse Kinematics of Robot Arm Using Neural Network”. English. In: *Robot Intelligence Technology and Applications 2*. Ed. by J.-H. Kim, E. T. . Matson, H. Myung, P. Xu, and F. Karray. Vol. 274. Advances in Intelligent Systems and Computing. Springer International Publishing, 2014, pp. 581–591. *Published*

- O. Ravn, N. A. Andersen, and T. T. Andersen. *UR10 Performance Analysis*. Tech. rep. Technical University of Denmark, Department of Electrical Engineering, 2014. *Published*

- T. T. Andersen. *Optimizing the Universal Robots ROS driver.* Tech. rep. Technical University of Denmark, Department of Electrical Engineering, 2015. *Published*

Acknowledgments

I would like to thank my supervisors Associate Professor Ole Ravn and Associate Professor Nils Axel Andersen for their continues guidance and support through my many years of studying all the different and interesting fields of robotics at DTU and for giving me the opportunity to work on this exciting project. They have also always been very helpful and open minded for discussing new possibilities and solutions.

I would also like to sincerely thank Professor Henrik Iskov Christensen for my very interesting research stay at Georgia Institute of Technology's College of Computing just as I would also like to thank all the RoboGrads from GT for both fruitful discussions as well as a very pleasant time in Atlanta. I especially owe Research Scientist Heni Ben Amor huge gratitude for his company, interest in my project and especially his very competent advice both throughout and after my stay.

The representatives from the other members of the innovation consortium, DIKU, Linco, IHFood, Butina, Robotcenter Danmark, Rose Poultry, Danpo, Danish Crown, Tican, and DMRI, I would like to thank for an inspiring collaboration, as well as for giving me the opportunity to work on some real world challenges.

The staff at the automation group I would like to thank for various assistance and good colleagueship throughout the years. I would especially like to thank Lisbeth Winter, Henrik Poulsen and Susanne Andersen for always being ready to help with matters large and small, as well as my office mate postdoc Haiyan Wu for many fruitful discussions.

Finally, I would like to express my gratefulness to my friends and family for their continuous support and help throughout this project.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis addresses the subject of using sensory information in a real-time manner to control robots. The real-time controlling constraint should be understood in the sense presented in [8]: "control[ling] an environment by receiving data, processing them, and returning the results sufficiently quickly to affect the environment at that time" and thus not in the classical computer science (CS) definition of a strict guaranteed timing constraint deadline.

1.1 Motivation and aim

Robots in all sizes and shapes are becoming more and more common in our society. From thinking of robots as something fairly dumb constrained to the factory floor, we now have autonomous public transportation systems, drones are taking to the sky for both recreational and military purposes, and entire warehouses are controlled by robots. There are surgical robots, cleaning robots, and farming robots. We have robots exploring the bottom of the ocean, the hostile environment of space, and nuclear disaster sites. Even robot prosthetics, robot pets, and robot competitions are no longer restricted to science fiction. In short, robots are spreading faster and faster into more and more domains.

The reasons for this spread are obviously many and often domain specific.

One of the multi-domain reasons are the increases in computer computational power, another the advances in sensor technology and processing algorithms. However, while all these advances have brought robotics forward at a blinding pace, the applications for the classical industrial robot does not seem to have grown significantly more advanced alongside this knowledge revolution. Although they have been optimized in terms of both accuracy, repeatability, speed, and maximum load, they are still mostly left to do one or more of the classical 4D tasks; Dumb, Dirty, Dull and Dangerous. And while it is certainly a good thing to have robots handle these kind of tasks, there seems no good reason to limit industrial robots to neither dumb nor dull and repetitious tasks.

One place with many repetitious tasks that would thus seem obvious to automate is in the food industry with processing of agricultural products. This is mostly labour intensive tasks with many workers at a production line doing the same repetitious task over and over again. But as it can be seen on figure 1.1 there is a very large natural variation in agricultural products and compensating autonomously for this variation requires quite a few sensors and real-time planning to handle. And getting robots to solve tasks using advanced sensors and a high degree of real-time planning in a reliable and autonomous fashion are often both difficult and expensive, and in an industrial setting reliability and automation are key requirements.

There are human operators monitoring and instructing the drones, the exploring robots, the transportation systems, the surgical and farming robots as well as the prosthetics. And the consequences of a failing pet or battle robot, or a badly or wrongly stacked pallet are usually acceptable.

But in a factory, where an trained robot operator is more expensive than most assembly line workers, where downtime can halt an entire production line, and where small inaccuracies can damage or destroy the end product, this is not an option. Before any factory owner would thus allow a robot to take over a task, be it simple or complex, he needs to be certain that the robot can do so both reliably and autonomously.

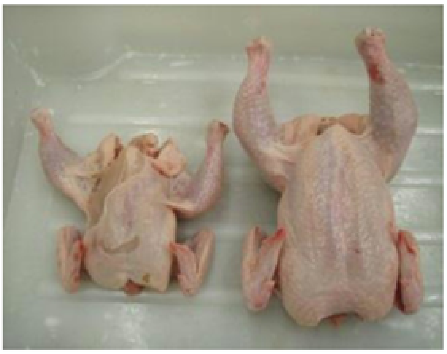

Figure 1.1: Example of the physical product variance in agricultural products

The motivation for and aim of the research done as part of this theses is to find ways which can improve this reliability and autonomous operation of robots while using sensors to handle non-uniform products. And while the emphasis and baseline might be on industrial robot manipulators, any such findings will obviously also be usable in advancing other areas within robotics. Therefore, I have not limited myself to only consider industrial robot manipulators, but am also looking into other major areas like mobile robots.

1.2 Reliability and automation

There are countless ways to increase the level of autonomous operation and reliability of a system, some of them are solution specific, while others can be applied in a more general manner.

The first included paper, paper A, presents a method for monitoring the execution of a task. By introducing an expert system, it is shown to be possible to increase the robustness of task solving by writing rules that defines the boundary of how a task should be solved. Besides monitoring the internal state of the robot and how the mission is progressing it can also monitor exterior conditions, and based on a knowledge of the entire environment both detect and handle arising situations and either prevent errors or at least help in ensuring a meaningful and autonomous degradation of operation.

As mentioned, one of the driving factors in the increased use of robots is the improved sensor technology. For combining exteroceptive sensors with robots and then getting them to relate their measurements to the robot system, a calibration is obviously a necessity. In paper B I present a method for calibrating a laser range scanner to a robot manipulator. The accuracy of the method is further improved in paper C.

Besides calibration, the methods presented in the two papers also gives a measure of the worst-case accuracy based on sensor resolution and -accuracy as well as the calibration target design parameters. In ensuring and documenting the consistency of a product, i.e. in relation to a ISO 9000 specification, it is very valuable to know that based on the calibration, the robot will perform to within a defined threshold.

Another very important aspect of reliable automation is accuracy. Positional accuracy has received a lot of attention, making robots far more accurate than humans will ever be. This is also reflected in the manufacturers' specification of all industrial robots, where both accuracy and repeatability is given. But temporal accuracy or reaction speed seems more like a complete unknown, with not even resellers or the manufacturers' customer support being able to supply this number. It is not difficult to imagine why, as it is simply not relevant for the current operations done by industrial robots where uniform objects makes it relatively easy to predict the required timing. However, for handling objects of varying sizes, which requires online planning, timing is crucial if the object is moving.

And with a trend in automation going towards using robots for smaller production batches and even single series products, knowing this number rather than measuring it in a static status is increasingly important.

In paper D I present a method for doing a rough mapping of this delay and then based on these mapping measurements, model the delay of the robot.

The importance and usefulness of knowing this delay is further elaborated in a separate chapter.

By covering these topics I hope to have made significant contributions towards better sensor based real-time control of robots. The scope of the title topic is so broad, though, that it is not realistic to hope to cover the entire field. My research is rather aimed at giving some tools that can aid in developing robust real-time control for robots based on sensory input with a focus on the main challenges identified as part of the innovation project "Real-time controlled robots for the meat industry". I will thus also use examples from and relate my results to automation of the meat industry.

1.3 Thesis structure

In the current chapter, I have presented the overall motivation and aim of this thesis and how it relates to my published research.

Chapter 2 describes what is meant by sensor based real-time control of robots and how it relates to the motivation and aim of this thesis. As temporal accuracy is an important part of achieving good sensor based real-time control, a large part of the chapter is also dedicated to elaborate on this topic.

In chapter 3 I present the actual work done in my published papers as well as conclude on what it adds to the problem at hand.

Chapter 4 will conclude on the thesis while the included papers have been reformatted and are included in the back as Paper A through D.

As the included papers and this thesis is the result of several years of work, it has not been possible to ensure a consistent nomenclature throughout all texts. Thus I use the expressions *industrial robot*, *industrial manipulator*, and *(industrial) robot arm* interchangeably throughout this thesis and the included papers. Unless otherwise explicitly stated, the expressions all cover the same general type of robot, namely the type of high precision robot arm commonly found in industry with 6 revolute joints including a spherical wrist and 6 degrees of freedom.

Chapter 2

Sensor based real-time control of robots

In this chapter I will explain what is meant by the expression *Sensor based real-time control of robots* and how it can contribute to the field of robotics. To establish a base to talk from, I will focus on industrial robots, but the conclusions is applicable to most other areas of robotics as well.

2.1 Typical robot operation in the industry

In the usual industrial setup involving robot manipulators used in industry today, and in the last decades, robot manipulators are placed along an assembly line where the product is moving along and manipulated by the robots one station at a time. A typical example of this can be seen on Figure 2.1. In some settings, the conveyor belt is moving while the robot interacts with the product, in others the line is stopped while the robot work. This is mostly defined by how accurate the work needs to be done and naturally what else is being done along the line. If the robot needs to spray paint a product, timing is not that critical as the nozzle can be opened before the product is at the robot and then the robot can continue to paint until it is certain the product has fully cleared the station. On the other hand, if timing is important for the accuracy, the product is usually stopped. This could be the case for screwing in a bolt which is quite difficult to do on something that is continuously moving, both for robots and for humans. Thus, such two operations should optimally not be done on the same continuous production line.

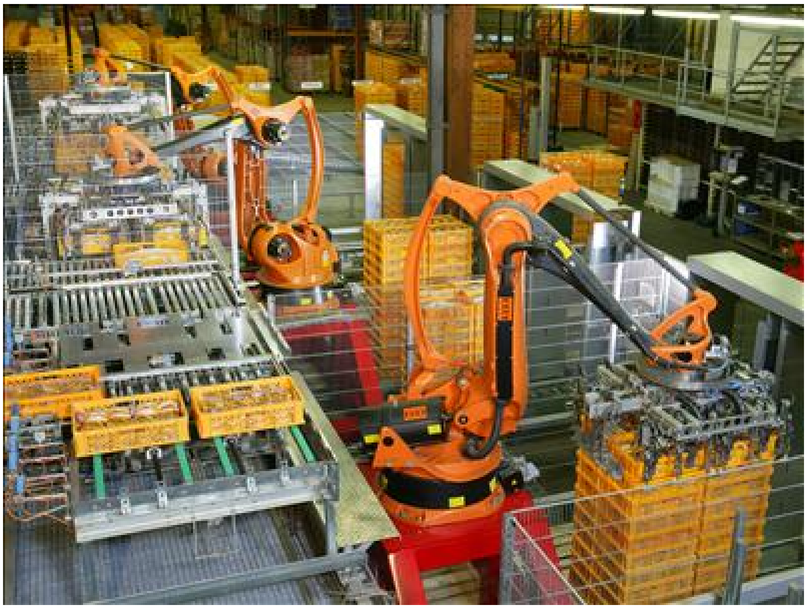

Figure 2.1: Typical industrial robot setup.

Regardless of the product motion, the robot is often using preplanned trajectories for solving its task, commonly referred to as offline operation. In the well-defined and static environment of an assembly line, it is not difficult to define the movements a robot should make to handle uniform products. With the product placed in fixtures so the pose is well defined, it is simply a matter of detecting when the object is at the station and the robot should start to execute its trajectories. As the starting time is determined by either starting before the product is at the station or stopping the entire line, timing is not an issue. Alternately, encoders can be used for synchronization. For a successful operation, it is thus only a matter of ensuring that the product is at the exact expected position on the conveyor belt. This is often the most difficult thing to ensure in pick and place operations, but the problem can usually be solved routinely by applying mechanical constrains like the before mentioned fixtures.

The fact that these solutions are dependent on the uniformity of the product and the assumption that the product is always at the exact same pose on the conveyor is clearly limiting what can be automated in the industry with an offline approach. The opposite is an online approach, where sensors are used to measure the shape, size, and position of the product, and based on this a trajectory is calculated in real-time. This is useful in industries where the above constraints does not hold true, for instance in the food industry with processing of agricultural products. Here, the natural variance of the products makes it impossible to use a predefined trajectory.

With agriculture being one of Denmark's largest export areas, any advances towards doing online control and thus being able to add more automation is especially important for the future competitiveness of Denmark. This is obviously true for many western countries where manual labor is expensive, and for that same reason, automatic solutions to different tasks have appeared all over the world over the last years where this project has been done. They are still quite few though, and still far from a turnkey solution.

Products that can handle unstructured positions, best exemplified with bin picking, have also started to appear on the marked in the last couple of years. They have yet to show good marked penetration, which also goes to show the difficulty in doing online planning reliably and at an acceptable speed.

Another trend that has been emerging within industrial robotics in the last couple of years is cooperation between humans and robots. Instead of building and designing the factory around the robots, the idea with cooperative robots, or cobots, is that they should cooperate with humans in a human environment using sensors for perception. One of the advantages of this is a more flexible production, because cobots is naturally better suited for changes due to their sensor based awareness of the surroundings as well as the fact that they usually don't need to be installed in an enclosure.

All these emerging solutions and trends thus relies on sensor based real-time control of robots.

Figure 2.2: ABB's cobot YuMi®.

2.2 Task phases

Regardless of the required operation, the robot task can usually be divided into three different phases: **cruise, approach** and **contact**. Note that not all robot setups necessarily goes through all phases, and in some cases, it can be difficult to establish exactly when the transition occurs. Below I give an overall description of the terms, knowing that it is in no way a complete description of all kinds of robot operation.

2.2.1 Cruise

Cruising is done whenever the robot needs to move from one pose to another as fast as possible and are thus usually planned in joint space. The first and last part of an operation is often done cruising, as the robot then gets ready to process the next product. Waiting idle is thus also part of this phase. The trajectory planning and execution is usually done without much input from sensors, except verifying that the working area around the robot is clear so the trajectory can be executed safely. This can be done very advanced using cameras, distance sensors and proximity sensors, or very simple with a cage. Many preplanned trajectories can be said to be done in the cruising phase and can usually be defined pretty naively.

2.2.2 Approach

The robot is in the approach phase when it is aligning itself with the product. The phase is triggered when the product is detected to be at a certain position near the robot, and is intended to bring the robot's tool close enough that it can start working on the product. In a 'dumb' setup where the assembly line is stopped before the robot starts handling the product, the stopping of the line usually triggers this phase where the robot then moves to a new pose under more strict constraints to avoid collisions. Thus defining these trajectories usually takes a bit more effort. In an advanced setup, visual servoing or other sensor-feedback based approaches could be used to bring the tool into the correct pose for manipulating the product.

2.2.3 Contact

The contact phase is when the robot is actually doing something to the product. This is when the robot closes its gripper and pick up the product but can also be when a painting robot is actually spraying or the current is flowing on a welding robot. Motion in the contact phase is usually highly constrained physically to ensure proper treatment of the product. For a pick-and-place robot, the constraint prevents dropping or damaging the product due to too high forces or moments, a deburring tool should not exert to much force on the product and a welding or painting robot should move at a correct and uniform velocity to ensure a good end result. This phase usually requires some calculations for proper execution, and is impossible to do robustly without sensors on non-uniform products.

2.2.4 Alternating between phases

From the above description it is clear that not all phases are used equally often. While a preprogrammed robot working at a stop-and-go assembly line will continuously change between the three phases, an optimal bin-picking robot will alternate between approach and contact for long periods of time and only change to cruising if the bin needs to be swapped with a full one or if the outfeed is full.

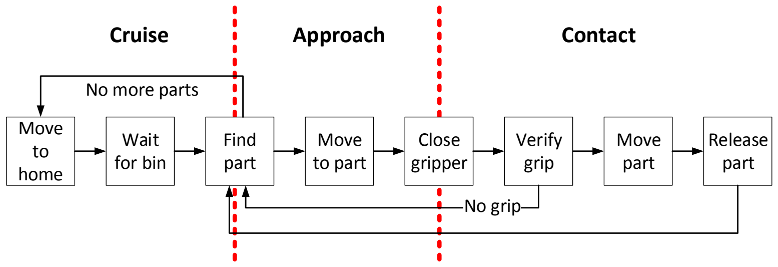

Figure 2.3: Example of tasks for bin picking and their phases

The sensors commonly used for online control in each phase also varies. For the cruising phase, the robot just needs to know that the area is clear, and so just like for the offline approach only LIDAR's, light curtains or other analog safety enclosure equipment is necessary.

When the robot approaches the product, it needs a sensor that can locate the product relative to the robot. If the shape and size of the product is known beforehand, monocular vision is sufficient. Otherwise, a sensor solution that gives 3D information is necessary. If the target is moving at a predictive velocity, which most products on a conveyor does, the data rate of the sensor doesn't need to be very high and 10-30 Hz will usually suffice.

As the robot gets nearer to the product and actually starts to manipulate it, higher data rate is often required to guarantee the accuracy. If the robot is physically manipulating the product, a force torque sensor (F/T sensor) could be valuable to make sure that the product is handled within the boundaries. This way a product can be moved at an optimal velocity relative to the product's mass.

As the required data input as well as the rate of data input changes between the phases, care has to be taken when changing between the controllers responsible for controlling the robot in each phase. The controller for visual servoing might be trying to move the robot in one direction while the impedance controller that uses the F/T sensor might want the robot to move in the opposite direction, leading to a very uneven transition between phases. Thus, some kind of adaptive controller like sliding weighting of the controllers' influence is needed for smooth transition and operation.

Even if data rates can be quite low in some of the phases, it is still very important that the delay from the data acquisition at the sensors to the actuation of the robot is low enough to make sure that the robot reacts while the sensed information is still relevant. For the safety-related input, this maximum reaction time is well defined and ensured for industrial robots. However, when it comes to reacting to the sensory information used to guide the robot in the approach and contact phases, the reaction time can vary quite a bit between different robots and even different control methods of the same robot. In optimizing performance in sensor based real-time control of robots it is therefore important to know the limitations imposed by the system components and their interfaces.

2.3 Interfacing between systems

When a robot needs to be interfaced to sensor systems like vision or a F/T sensor, some robotic manufactures sell readymade subsystems made for seamless integration of sensor based solutions. This has the advantage of relatively easy deployment and a single supplier of support. It is also fair to assume that this offers the best performance attainable with that particular robot.

The disadvantage of this approach is the increased cost associated with the vendor lock-in and the limited applications. If the offered solution does not fit, it can be very difficult and expensive to get it customized.

As the robotic manufactures obviously can't supply solutions for all industries and all use cases, most manufacturers of industrial robots provide instructional languages for their robot systems [9]. This way both researches and professional implementers can develop custom solutions to non-trivial and advanced tasks, while still maintaining a fairly high performance.

This approach still has the disadvantage of vendor lock-in due to the usage of proprietary instructions. If an implementer has a bin-picking solution with a code base that is optimized for one kind of robot controller and arm with a specific type of camera, it might be difficult for sell that solution to a company that only uses another brand of manipulators and cameras.

And as it is noted in [9], the instruction set is tailored to the specific robot controller and thus only offers a fixed set of instructions, causing even the robot manufacturers difficulties adapting the controller to modern requirements. Furthermore, if one of the products used in the solution reaches its end of life, the entire code base might need to be re-written as joint space commands or sensor parameters can hinder portability. Finally, it also requires a good knowledge and understanding of the individual manufacturers' interfaces for optimal performance.

To remedy the issue with vendor lock-in, more and more solutions are based around a middleware. This can be thought of as a layer that exist between a computer's operating system and the programs that are executed on the computer. The main purpose of the middleware is to allow easy communication between processes running on a computer or even processes distributed on many computers. This makes it possible to have one program that interfaces to the robot, another program that interfaces to the sensor, and a third program that reads the output from the sensor program and calculates an appropriate trajectory, which is then written to the robot control program. Obviously, this can be further modularized, like having separate programs for calculating grip poses and trajectories.

The advantage of this approach is that it is possible to change only one part of the system, like the robot, and its associated driver and kinematics solver, without modifying anything else. The disadvantage is that now the developers also have to develop and maintain a middleware, while still having to be experts on the interfaces of all the supported products. This problem can be solved by distributed development, where each robot and sensor manufacturer supply the program required to use their own problem. However, as the manufacturers have a large interest in vendor lock-in, especially in the robotics area where a few big companies dominates the marked, such an agreement has not been reached. Moreover, if it was possible to agree amongst the manufacturers on such a middleware, it would be more natural to define a common interface and thus the need for middlewares would not be present in the first place.

Seeing that there is a need, there have been several attempts in the robotics research community at creating a good robotic middleware. In a 2012 study[10] the authors analyze 20 different robotics middlewares and goes to list further 13. Three years later, at the time of writing this thesis, only 12 of those have been updated within the last 2 years though. Another 9 seems to have vanished and are only referenced in research articles, while the last 12 still maintain a homepage, but no changes to code nor any updated has been posted for more than two years.

Of those that have made it, one of the most popular choices of robotic middleware seems to be the middleware called Robot Operating System (ROS)[11]. It has users and active contributors from all over the world and has interfaces for several of the other middlewares from [10] that are still active like Stage[12], Webots[13] and Orocos[14]. It seems to have reached a critical mass of maintainers which can prevent it from disappearing into the unknown like many others before it, which were often developed, maintained, and in some cases only used, by a single company or university group.

ROS is spreading quickly in the research community and at the time of writing, where ROS just turned 8 years, the introducing paper from IEEE ICRA 2009[11] has more than 2.000 citations according to Google Scholar. Moreover, for the last four years running, it has even had its own conference ROSCon back to back with either ICRA or IROS, with attendance of both researchers, users, and large established companies like NVIDIA, Qualcomm, BMW, Intel and more. In addition, in 2015 alone, more than $150 million in venture capital was invested in business utilizing ROS.

ROS has even reached a size where robot manufactures like Motoman, Rethink Robotics, Intel, and Clearpath Robotics are actively participating in developing ROS interfaces for their products, while the ROS-Industrial initiative are developing and maintaining interfaces for many other industrial robots.

When chaining different pieces of hardware together, and the controlling software is executed on stand-alone computers rather than on the robot controller, performance is likely to suffer due to communication delays. Delays that are inherently unavoidable in distributed systems like those utilizing a middleware. A simple example of this is sketched in Figure 2.4 and explained in the following section.

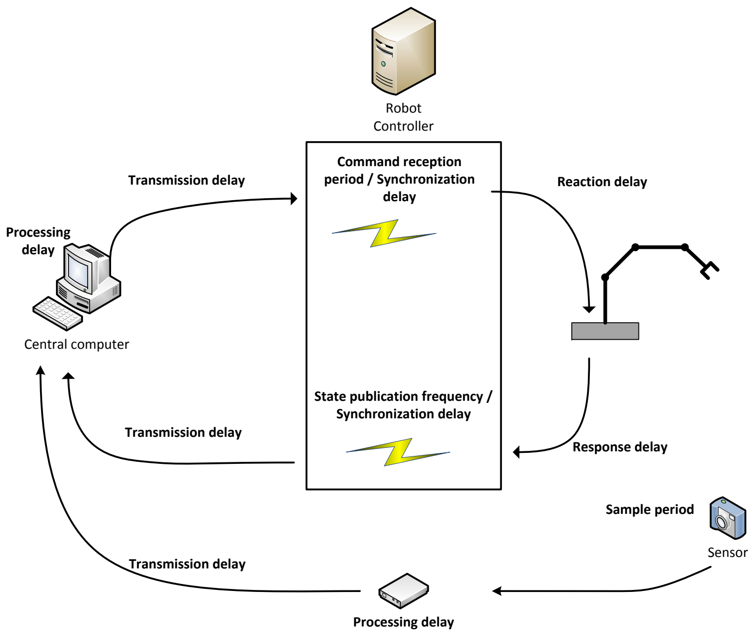

Figure 2.4: Chart of the different delays in a distributed robotic system

2.4 Delays in distributed robotics systems

Knowing the delays is important in optimizing the performance, and in some cases even to determine if a certain setup is physically feasible.

Imagine a setup where a robot needs to pick up something fragile where the size and position is unknown. In the approach phase the robot moves towards the product, and as it touches the object and thus transits to the contact phase, the robot should stop within 0.1 mm to avoid risk overturning, rolling or damaging the product. This could be picking up an egg or a live chicken, but this sort of feedback-based accuracy is also necessary for cutting with a knife along a bone.

While eye-in-hand visual servoing is used for most of the approach phase, it is not possible to use in the final moments before contact due to self-occlusion and an eye-to-hand vision solution is unable to give the required sub-millimeter accuracy. As visual servoing is not a viable solution for the final approach a F/T sensor is used to detect when the robot touches the product. The F/T sensor is sampled every 0.5 ms, while the robot's control cycle is executed every 8 ms. These numbers are usually easy to get from the performance sheets of most robot and sensor hardware.

If this is all done as part of a manufacturer's closed system, it is fair to expect that the controller reads the robot state, then the F/T sensor, and based on this calculates whether the robot is in contact with the object or not. The worst-case scenario would be if the robot comes in contact with the product just after the reading used for the next control cycle. In that case it will take up to 8.5 ms until the robot reacts to the contact and stops. Furthermore I assume 1.5 ms to handle transferring data from the sensor and calculating the desired control signal, in total 10 ms. Thus the robot will be able to move at up to 10 mm/s at the final approach towards the product and still react to the contact within the defined 0.1 mm. Note that the robot is only specified to be signaled to stop within that boundary and thus I do not consider the actual deceleration in these calculations.

When a robot middleware is used, the same degree of synchronization cannot be expected. Any distributed architecture also needs to be considered. It is thus valuable to have a map of the individual delays and how they can affect the worst-case performance of a system.

When something is in the range of a sensor to be sensed, the corresponding signal is not transmitted from the sensor instantaneous due to the need for digitizing the signal in A/D converters so a computer can process it. This delay can be as big as the inverse of the sensors sampling period, i.e. up to 0.5 ms for a F/T sensor with a 2 kHz sampling period. The values from the A/D converters is then gathered and packed in a specified communication protocol for other units, like a computer, to read the data from the sensor. This causes a processing delay in the sensor. Actually transmitting the data from the sensor to the central computer running the middleware further introduces a transmission delay. This is the lower part of Figure 2.4.

Once the sensor data is at the central computer it needs to be processed to determine an appropriate instruction to the robot manipulator, which should then be transmitted to the robot's controller. This causes yet another processing delay followed by a transmission delay.

When the instruction is at the controller, it has so wait for the controller to actually read the instruction. This is usually done as part of the control cycle running at a certain frequency, i.e. 125 Hz for a Universal Robot (UR), which means the synchronization delay can be up to 8 ms.

Once the robot controller has read the instruction from the input, it needs to parse it and convert it into commands for the individual joint motor controllers. These in turn needs to energize coils, overcome gear backlash, release breaks and build up torque to overcome the dynamics of the system, before the robot physically reacts to the instruction. This is denoted as the reaction delay on Figure 2.4.

As part of the control cycle, data about the robot state like joint position and -velocity is also transmitted back to the control computer. Just like with the external sensor it takes some time to measure and convert this. The robot state data is transmitted at a given frequency. The time from when data is measured until it is actually transmitted introduces another synchronization delay. Packing the data according to the communication protocol takes some time and should be completed before the real-time scheduler signals that it is the right time to publish the data. This is the source of this synchronization delay. The total measurement and synchronization delay is named response delay on Figure 2.4 as there is no way to distinguish the two from each other. Transmitting the data from the robot controller back to the central computer introduces yet another transmission delay.

The size of each delay will often vary depending on the hardware as well as the physical setup. In [4] I found that with the Kuka Robot Sensor Interface(RSI)[15] the actuation delay averaged around 88 ms, while in [7] I found that using ROS with the python based driver, that has been the standard in ROS for several years for the UR, the actuation delay is around 170 ms.

Using the UR in the example I started this section with, the performance of such a system would drop significantly. An assumed 2 ms combined delay from sampling the F/T sensor, transmitting the reading, processing it and then transmit it to the controller would, together with the worst case synchronization delay of 8 ms and the actuation delay of 170 ms, result in a total delay of 180 ms. This drops the maximum velocity of the robot to just 0.56 mm/s, equivalent of a performance drop of almost a factor 18.

If the self-occlusion occurs when the robot is 2 cm away from the object, the robot would take 1.7 s to get to the product in the first example. When using ROS together with the current python based driver, it would instead take the robot 36 s to get to the product. The business case for a robot solution where the robot uses more than half a minute to pick up a single product is significantly worse than spending only a few seconds. Thus, this example goes to show that knowing the delays in a system beforehand could save quite a lot of money in developing a solution that in the end is not feasible.

2.4.1 Developing a more efficient driver

As explained above, the python ROS driver for the UR is not very suitable for sensor based real-time control. However, based on the work we did in [6] I believed a better driver could be developed for the URs.

Further tests with the proprietary instruction set for the UR showed that with joint positional instructions, the controller buffered some commands to better predict and plan the trajectory. The downside of this is an increased reaction delay, up to 124 ms. With the newest firmware it is possible to reduce the lookahead and thus the reaction delay to around 40 ms at the expense of a less smooth joint velocity profile.

The joint velocity instructions showed better performance, as the controller executed these commands in the first command cycle after the command was received by the controller, making the response delay merely a single control cycle of 8 ms.

Furthermore, a multithreading strategy was used on the controller with two threads. One thread was synchronized with the robot controller running at 125 Hz and in each control cycle, the joint velocity command is called with velocities set by global variables.

The other thread continuously read data from a socket input. This data is used to update the values of the global velocity variables. This makes it possible to do a sort of inverse oversampling, where the controlling variables can be updated several times before they are actually used. It is thus possible to use sensors with a sample frequency higher than that of the robot and use every sample to compute an updated control signal for the robot while ensuring that the most recent sensor reading is always used to control the robot.

The entire process of designing the driver as well as performance testing different control strategies are thoroughly described in [7]. The presented method makes it possible to get the actuation delay of the UR down to just 8 ms and the synchronization delay down to at most 1 ms. The synchronization delay could probably be lowered further, but this is mostly limited by network performance.

A plot comparing the performance of the two different drivers can be seen in Figure 2.5. The trajectory was executed 10 times with each driver, and the resulting 20 position graphs are shown in the figure. As it can be seen, the line for the old driver is slightly wider than that of the new driver, which goes to show that the new driver also performs more uniformly, as well as reacts much faster.

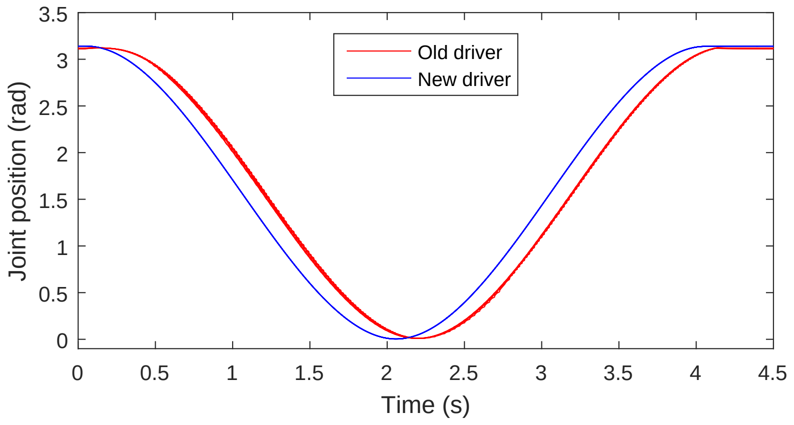

Figure 2.5: Comparison of the performance of the two different drivers. Note that there is 10 lines for both drivers, which explains why the line for the old driver looks slightly wider.

The old driver also has a small steady state error, causing it to not quite arrive at the target position. This can also be seen in the plot where the old driver at first moves a bit in the wrong direction, as the initial position from the previous trajectory was not perfect, and by the fact that the two lines does not end in the same position. They were both supposed to end in the position where the blue line is.

The total worst case delay from my previous example with the F/T sensor would, with the method presented in [7], drop to 11 ms and the maximum velocity would be 9.1 mm/s, only a 9% decrease compared to the assumed optimal manufacturer method. Thus, neither ROS nor UR can be said to be the cause of the found infeasibility in the previous example.

The infeasibility is rather caused by a suboptimal driver. The facts that such drivers exist, and the measured delay of a Kuka robot in [4] which does not use a middleware, only goes to support the argument that it is important to determine the delays in robotics systems.

Based on this, it would also be worth to consider if other drivers could be improved with the multithreading strategy presented in [7].

2.5 Measuring the delay

To see if there is actually anything to gain from optimizing a driver, it is necessary to know the delay. To that end I presented a method for measuring the delay in robot actuators in [4]. By mounting an accelerometer or a gyroscope on the robot it is possible to measure exactly when the robot physically moves, where the accelerometer is used for prismatic joints and the gyroscope is used for revolute joints. A computer is connected to both the sensor and the robot and a program on the computer then starts streaming speed commands to the robot and notes the time when the stream was started. State data from the robot and measurements from the sensor along with timestamps for when the data is received are then logged until the robot finishes executing the command. Based on the logged data it is then possible to find both the actuation and the response delay. A plot of such a measurement is shown in Figure 2.6.

It is important to remember the other delays in the system and attempt to either compensate for those or at least account for them in the data. The sensor used for the measurement trial shown above measured at 8 kHz which is also the explanation to why the blue line in Figure 2.6 has many more data points than the other two lines. Thus measurement could depict the situation up to 0.125 ms late. The magnitude of that delay is negligible compared to the actuation delay of almost 100 ms and the response delay of roughly 20 ms.

Techniques like using sensor interrupts for timing, precomputed trajectories, logging to memory instead of disk and direct network connection between computer and robot controller was used to minimize the inherent measurement system delay.

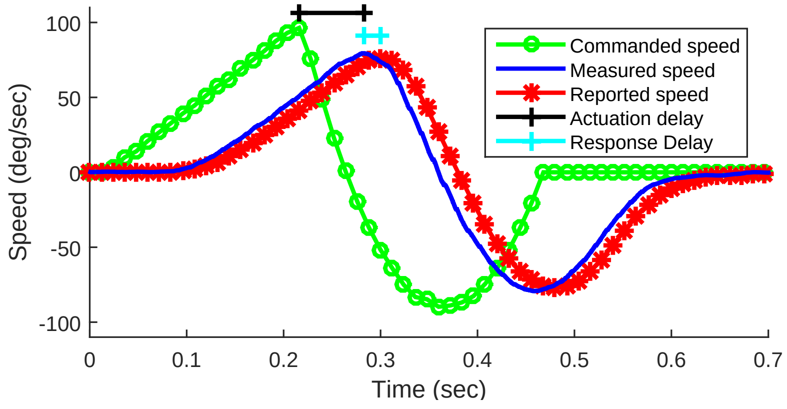

Figure 2.6: Plot of logged data from a delay measurement. Reprint of Figure D.5.

Based on the logged data, three timestamps can be compared: When was a command for a given velocity issued (green graph on Figure 2.6), when was the robot actually moving at that velocity based on physical measurements (blue graph on Figure 2.6), and when did the robot state data reflect that the robot was moving at the speed (red graph on Figure 2.6). The difference between the first two gives the actuation delay, while the difference between the latter two gives the response delay. As the received robot state is used to trigger when to send the next command in the stream, the sum of the two delays will always equal an integer multiple of the robot's control period time.

Based on the experiments done as part of [4], it was found that both the Kuka KR 5 Sixx and the UR10 have a constant delay throughout the trajectory, but the delays are varying relative to the robot's joint configuration when starting the trajectory. It is thus necessary to measure the delay with the robot starting in all the different joint combinations possible. Due to the power of exponentials, the number of different joint combinations and thus the time it would take to complete such a complete mapping makes it infeasible to do for a robot with more than 1 or 2 joints. Instead in [4] we presented how a model could be made based on sampling a subspace of the joint configuration combinations and use that to train a machine learning algorithm.

Based on that model, it is possible to define a function $d(\mathbf{j}, n, s)$ which returns the actuation delay of a robot's joint n, when the robot is in joint configuration $\mathbf{j}$ and moving in direction s. $\mathbf{j}$ is a vector with as many elements as the robot has joints. Likewise another function $r(\mathbf{j}, n, s)$ can be defined which returns the response delay.

2.6 Using the delay models

Determining whether the performance of a driver can be increased is only a part of what a delay model can be used for, and certainly a complete model is seldom necessary for that as the actuation delay usually doesn't vary more than a single control cycle. In the previous example with the F/T sensor, the limiting factor was the reaction *speed* of the robot in the transition between the approach and contact phase. Therefore, while knowledge about the magnitude and nature of every delay in a system can be used to predict whether something is possible with the given system, it might not be possible to do anything about it, as a physical system will always exhibit some delay. However, knowledge of the delay is also useful when temporal accuracy, or the *timing* of the reaction, is important.

Temporal accuracy is thus required whenever the robot needs to interact with something that is moving relative to the robot.

Measuring and modelling the delays are covered in [4], but not how to actually use the model. This is covered in the following text.

2.6.1 Forward temporal prediction

With forward kinematics, it is possible to determine where a robot's joints will be in Cartesian space, given some coordinates in joint space. Likewise, it is possible to use forward temporal prediction to estimate when a robot's joint will actually be at a given position based on a fully defined trajectory. A fully

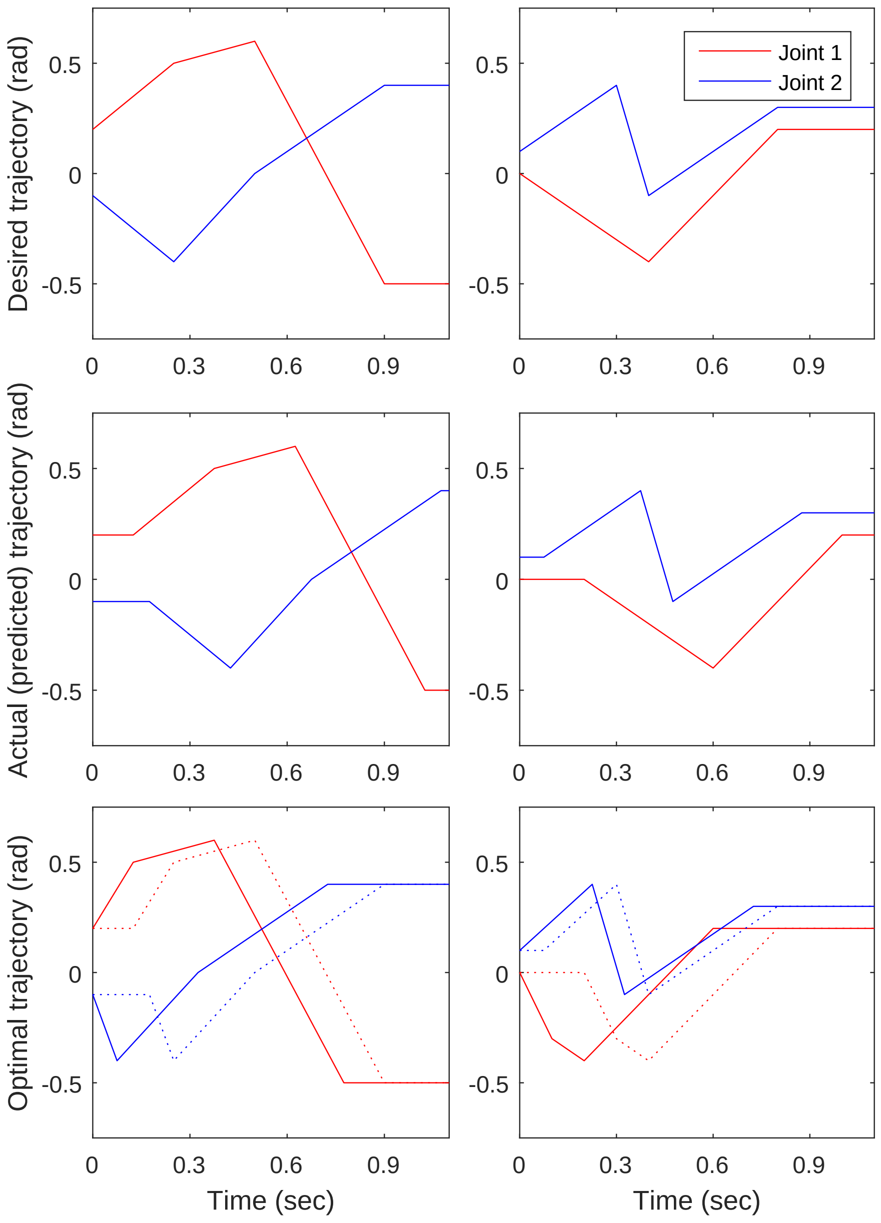

Figure 2.7: Two (left and right) example trajectories. Top row: Desired positional trajectory. Middle row: Actual predicted positional trajectory based on forward temporal prediction. Bottom row: Optimal positional trajectory based on inverse temporal prediction. The dotted lines are the predicted resulting positions.

defined trajectory has a starting position for every joint, one or more target positions for each joint, and on or more durations specifying how long after the trajectory is started that each joint should be at their target positions.

Two example trajectories of a two joint robot can be seen in Figure 2.7 top. At the robot's current configuration, joint 1 has a delay of 125 ms when moving in the positive direction and 200 ms when moving in the negative direction. For joint 2, the delays are 175 ms and 75 ms, respectively. These values are purposely chosen relatively high to exaggerate the effect of the delay on the plot.

As the delay is constant throughout the trajectory, it is possible to do forward temporal prediction by first determining the starting direction of each joint s and then use the function $d(\mathbf{j},n,s)$ to get the delay of each individual joint. This delay is then added to the durations in the trajectory.

The resulting trajectory with a temporally accurate estimate of the joints positions can be seen in Figure 2.7 middle. As it can be seen, the trajectory is in practice translated along the time axis corresponding to the delay of the joints.

Forward temporal prediction is thus used after the trajectory is planned and calculated to comply with all constraints, to accurately predict when the joints will be at what position once the trajectory is transmitted to the controller. Forward temporal prediction is therefore useful when an external process needs to be synchronized with the robot's motion.

2.6.2 Inverse temporal prediction

Just as inverse kinematics is used to determine the joint positions that leads to a desired Cartesian position, inverse temporal prediction is used to determine the exact duration of a trajectory if the robot must be at a certain position at a specific time. Again, a fully defined trajectory is needed.

Reusing the trajectories from Figure 2.7 top, it is thus possible to use inverse temporal prediction to make sure that the robot actually arrives at the target positions after the desired duration.

The call to $d(\mathbf{j}, n, s)$ is used again with the same parameters, but for inverse temporal prediction the delay is subtracted from the duration the robot should take to get to each target position. Note that the duration is defined as the time from the start of the trajectory, not the time from the previous point in the trajectory. The resulting optimal trajectory is shown on Figure 2.7 bottom. The dotted lines show the planned optimal trajectory, taking the actuation delay into account. As it can be seen, the robot actually arriving at the desired moment. This is done by compressing the first part of the trajectory, resulting in higher velocity of the joints.

Inverse temporal prediction is therefore used before the final trajectory is planned to ensure that the joints will be at the target positions after the specified time has passed since the trajectory was transmitted to the controller. Inverse temporal prediction is thus useful when the robot needs to synchronize its motions with an external process.

2.6.3 Current state correction

The current robot state that the robot controller transmit back to the central computer also has a small delay as previously explained. Thus, the state data actually describes a state in the past rather than the present. How old the state data is can be determined with the $r(\mathbf{j}, n, s)$ function.

Once the magnitude of the delay is determined, the current state can be corrected by extrapolating based on the state data and previous commands to the robot. When using the previous commands to extrapolate, it is important to take both delays into account.

While the two prediction methods is used in the planning and timing phase, the current state correction can be used to find the true current position, velocity, and acceleration of the robot when used as input to a control loop and thus improve the performance of the robot by limiting overshoot.

2.6.4 Improving performance with a delay model

As it was not the reaction timing that limited the previous example with the F/T sensor, I will instead consider an example where the robot has to pick something up from a conveyor belt moving at 0.5 m/s. This could be a cut off piece of meat placed by the butcher semi randomly on a conveyor. Using its sensors, the system know where the object is on the conveyor belt and with the knowledge of the conveyor velocity it determines that the object will be at an ideal position for grasping in 5 s. To minimize the stress of the robot's joints, the naive planner thus calculates a 5 s trajectory to move from the robot's current position to the ideal grasping position.

The Kuka robot from [4] is used, with its average actuation delay of 88 ms as it can be seen on the rightmost box on Figure 2.8. If the system is unaware of this delay and uses the 5 s trajectory to get to the product, the robot will then on average arrive 88 ms, or 44 mm, behind where it was supposed to grasp the object.

Knowledge of the delay could help to reduce this error significantly. As the robot needs to synchronize to an external object and be at a certain position at a specified time, the inverse temporal prediction method is used. The average error when using the learnt model based on Gaussian Processes is 3.06 ms [4], which would decrease the average positional error to only 1.5 mm. This is an improvement of more than a factor 28.

Mapping and modelling this delay takes quite some time. The 33,500 trials on the Kuka in [4] each took roughly 3 seconds, and two of the six joints was not mapped. Furthermore, the training of the machine learning algorithms also takes quite some time. It is thus worthwhile to consider only to perform a single trial and use that delay as a basis for the prediction.

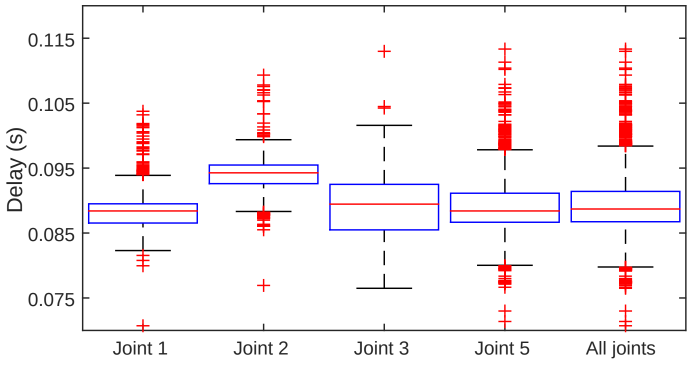

Figure 2.8: Boxplot of individual joints' actuation delay and a combined for of all joints of the Kuka 5 Sixx. Reprint of Figure D.8 left.

As it can be seen on Figure 2.8, 99.3% of the actuation delays, indicated by the whiskers on the boxplot, varies between 76 ms and 103 ms, depending on which joint is moved and where the joint start position is. Statistically a single measurement would be on the median, represented by the red line. Thus using the median actuation delay of 88 ms could result in an error in the range of 12 ms too early to 15 ms too late, or between 6 mm before and 7.5 mm after the object, moving at 0.5 m/s.

The average error of the median is only 3.35 ms though[4], resulting in an average positional error of 1.7 mm. This is still an improvement of more than a factor 26, but 9.4% worse than using the full model.

Thus, it is not possible to say anything conclusive as to whether it is necessary to do a full mapping, or a single measurement per joint will suffice. The single measurement could be used to determine if there is a significant delay though, and thus aid in the decision as to whether there is anything to gain by compensating for the delay. However, with only a few measurements, it is not possible to say anything about the spread of the delays. In the end, the decision depends on the use case and the sensitivity of the application to delays.

As I have shown in this chapter, delays have a big impact on sensor based real-time control of robots though. This was demonstrated through two different examples using data from two real world robots.

In the first example I showed how a suboptimal driver for the UR made real-time control with a F/T sensor infeasible due to low reaction speed. With the knowledge of the high actuation delay and a method for measuring how different control strategies affected the delay, I could make a driver[7] with an actuation delay of only 8 ms which improved the performance significantly. Furthermore, it is possible to send commands to the controller almost arbitrarily fast, making the limiting factor of the synchronization delay how fast commands can be calculated and transmitted from the central computer to the robot controller.

In the second example, I showed how knowledge of the delay in a system could be used to synchronize the motion of an externally controlled robot with its surroundings. Using a Kuka 5 Sixx robot controlled with RSI introduced an average delay of 88 ms which uncompensated could lead to an error of 4.3 cm when trying to grip something moving at 0.5 m/s.

By measuring and modelling the delay, it was possible to use inverse temporal prediction to lower the average error to only 0.15 cm.

Chapter 3

Published research

In this chapter I will present my published findings and explain how they each contribute towards more reliable sensor based real-time control of robots.

3.1 Real-time task monitoring

When decisions have to be made and plans calculated in real-time based on noisy sensory input, several potential things could go wrong with every object that is handled. Machinery could stop responding, external disturbances could affect sensor readings, computer programs could crash, sensors could break or saturate, foreign object could make it into the robot's working area and so on. The presence of potential failure modes makes it necessary to do real-time monitoring of the task execution.

In [1] I presented a method for detecting exceptions and failures of tasks as well as how to handle these issues as part of mission control. In order to have a real-time system with access to all system parameters I used DTU's Small Mobile Robot (SMR) in the research, but the work is equally applicable to other kinds of robots.

This was accomplished using an expert system called Jess which I connected to the MobotWare framework[16], so that all the robot's sensory input was available to the expert system as facts. Likewise, all the results of processing input data was also available as facts. A system overview can be seen in Figure 3.1.

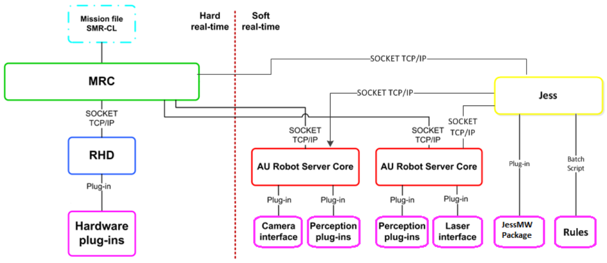

Figure 3.1: System overview of MobotWare and Jess. Reprint from [1]

Using the expert system, I showed how rules could be defined that considers the entire system state by asynchronously evaluating all facts. The rules was used to both do sanity checks on input and results as well as to plan how to successfully accomplish a mission by solving a set of tasks.

For each task potential exceptions was identified and strategies to identify and handle these was defined and formulated as rules.

It was shown that the rule-based system could successfully solve a complete mission this very very robustly. When the robot ended up in erroneous states, it was able to handle these situations on its own and ensure that the mission was successful.

As an example, the robot autonomously handled being stuck at one point. The robot's navigation system planned a path for the robot and wanted it to drive forward. Suddenly an opening door blocks the path and the low-level protection system stops the robot. Although the robot is unable to see behind itself, the expert system knew where the robot came from and thus that the path behind it was clear, so it made the robot reverse. From here, it could ask the navigation system for a new plan towards the mission goal.

In another situation, the laser based localization system lost its orientation and thus the robot started heading in the wrong direction. The expert system noticed that the camera detected landmarks that it did not expect and could conclude that the localization system was failing. The expert system then informs the navigation system of the correct pose of the robot, which makes it possible to safely navigate to the target position even though some subsystems fail.

These examples shows how task monitoring can aid robustness in sensor-based robot solutions. Even in situations where it is not possible for the robot to solve its task, i.e. if the robot is stuck or an actuator stops working, the expert system can alert a human operator with a meaningful description of the problem to ensure that the problem is eliminated as fast as possible.

The exception detection and handling can easily be used in other fields within robotics like industrial robots. Besides doing sanity check on sensory input and processing results, the expert system can also continuously monitor the execution of the robot's mission. If the robot for instance needs to pick up an object from a conveyor belt and place it at another station, the expert system could easily detect if something went wrong.

From the gripper's opening distance it might deduce that the grip quality if insufficient and determine that there is time enough to try another grip pose or let the object continue down the conveyor to another robot, also monitored by the same expert system.

The system would also be able to use force sensors in the robot, or maybe even motor current readings, to detect if the robot accidentally drop the object. It can then stop the robot and make it return to the conveyor to pick up the next object instead of wasting time on completing the motion to place an object, which is no longer grasped by the robot. The expert system could also call a human worker, or another robot, to come and pick up the dropped object. Furthermore, the system could detect if a robot is starting to drop a lot of objects which could signal that the gripper surfaces are worn and need replacement.

3.2 Robot to laser calibration

Whenever food is handled, high attention to hygiene is required. For the production of meat products, this translates in to daily thorough cleaning of the entire production facility. A part of this cleaning is washing all surfaces using high-pressured water cleaners.

For one of the collaborating partners in the "Real-time controlled robots for the meat industry" project, this cleaning process sometimes shifted the orientation of their sensors, which in turn led to a suboptimal yield of the handled product. As the recalibration was a manual process requiring high knowledge of the system, the implementer's engineers were often required to travel across the country to perform the recalibration.

In [2] I presented a method for autonomous translational and rotational calibration and recalibration between a laser scanner and a robot manipulator to help alleviate this problem in general. Unlike previously published methods, the presented method does not require other pre-calibrated sensors nor intensity measurement from the laser range scanner.

I show how a calibration target can be optimally designed taking into account the often confined space in a production environment. The design analysis results in a flat two-finger calibration target with a varying distance between the two fingers. Unlike the target in other published methods, the flat target used in this paper can easily fit between the sensor and a conveyor belt. Once the initial calibration is done and the method is used for re-calibration where the system has a good guess of the transformation between the robot and sensor, the size of the target can be further reduced.

Using this target, I show how the transformation between an industrial robot manipulator and a laser range scanner can be found using only a single scan in each of two different positions. In each scan, the pose of the target can be determined, and with the knowledge of the robot's motion between

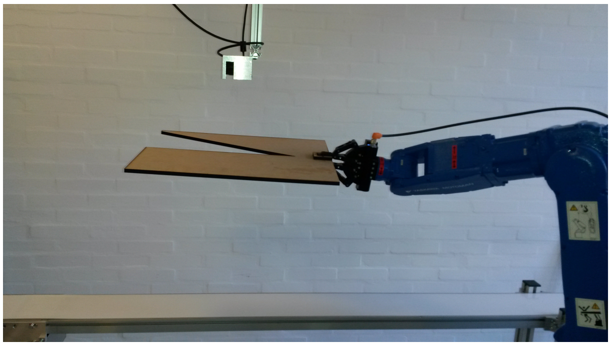

Figure 3.2: Test setup with up-scaled calibration target. Reprint from [2]

the two positions, the transformation between the robot and the sensor can be calculated using simple trigonometric.

Unlike previously published methods, I furthermore showed how the worst-case calibration accuracy could be determined based on calibration target size and the laser scanner characteristics. Besides for comparing performance of different methods, this information can also be used to document production consistency, i.e. for an ISO9000 certification.

In the paper we find that the translational calibration accuracy can be optimized by averaging over several measurements from the scanner, giving a worst-case error of 1 mm along all axes. The rotational calibration accuracy depends heavily on the angular resolution of the scanner and could thus quickly degrade the quality of the calibration. With an angular resolution on the laser scanner of 0.35°, the angular error in the calibration could be larger than 7° around one of the axis.

Thus in [3] I presented an optimization of the previous method. By using only one or two laser rays from the laser range scanner and exploiting the accurate nature of most industrial robot manipulators to move the target's edges into these rays, it is shown that the negative effect of the laser scanner's angular quantization can be annulled.

Table 3.1: Comparison of performance between the two scan method from [2] and the single beam method from [3]. K is number of scans when measuring the distance from the scanner to the target. Reprint from [3]

	Two scan	Single beam
X_L (mm)	±1	0 for $K \to \infty$
Y_L (mm)	±1	±0.006
Z_L (mm)	±1	±0.006
Rot_{X_L} (°)	±7.07	±0.0097
Rot_{Y_L} (°)	$\begin{cases} +2.64 \\ -0.53 \end{cases}$	±0.0164
Rot_{Z_L} (°)	±6.8	0.00172

The target from [2] is reused, just as a method for determining the worst-case calibration accuracy is also presented in this paper.

By moving the target around so that the individual laser ray hit the edge of the target, it is possible to determine the pose of the target much more accurately. As I showed in the paper, it is possible to find the edge of the target with an error of less than 6 μm, using a cheap Hokuyo URG-04LX laser range scanner and a Motoman MH5L which is a standard 6R industrial robot manipulator. A plot of measurements done while moving the target at 1 μm increments can be seen in Figure 3.3.

With the improved method, it is shown that the worst-case accuracy is greatly improved. A comparison between the two methods performance can be seen in Table 3.1. It should be noted that these results are theoretical and is only verified with respect to the characteristics of the laser scanner. The robot arms ability for micrometer movements as well as the production accuracy and measurement of the calibration target will also influence the worst-case accuracy. This is true for all kinds of calibration, though.

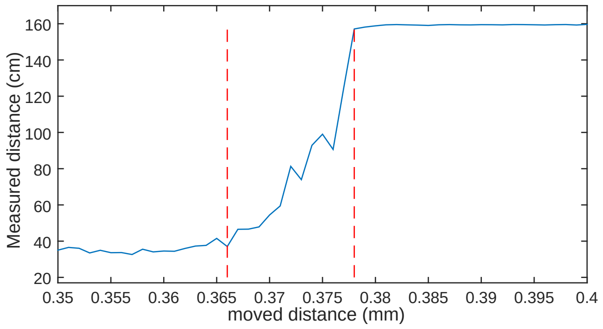

Figure 3.3: Distance measured by laser scanner while moving target underneath in 1 μm steps. The red lines denotes a 12 μm interval. Reprint from [3]

Both methods is also applicable with both 2D and 3D laser range scanners and all robots with 6 degrees of freedom.

3.3 Modelling robot delay

In the previous chapter, I described how useful knowledge of the delay of a robot manipulator is for doing sensor based real-time control of robots. I also described how a gyroscope or accelerometer can be used to measure this delay to map the delay profile of a robot.

Measuring and mapping the entire delay profile of a robot is very time consuming though. The delay has to be measured for one joint at a time, and as the delay depends on the actual joint configuration and the direction of which the joint is moved, the number of combinations of joint configurations quickly makes it infeasible to measure the full delay profile.

Along with publishing the method for measuring the delay in [4], I therefore also showed that it was possible to model the delay based on a incomplete mapping of the delay at only a few selected joint configurations.

The modelling was done with machine learning algorithms, so the same method can be used on different robot platforms, instead of having to develop models for each kind of robot platform.

To further try and see which machine learning algorithm works best at predicting the delay, three popular machine learning algorithms was tested; Neural Networks, Regression Trees and Gaussian Processes.

As learning parameters for the models, the position of the robot's joints, which joint to move and which direction the joint move are used. As it can be seen in Figure 3.4 the joint configuration of the robot influences the delay of the robot. This influence is assumed to be caused by the external forces of gravity that the robot has to overcome.

A subset of all the delay measurement was randomly chosen and distributed into 10 bins. These bins were used for k-fold cross validation, where nine of the bins where used to train the model and the last bin used to test the model's performance. The mean square error from each fold was then averaged and used as a measure of how well the models could predict the delay. This was

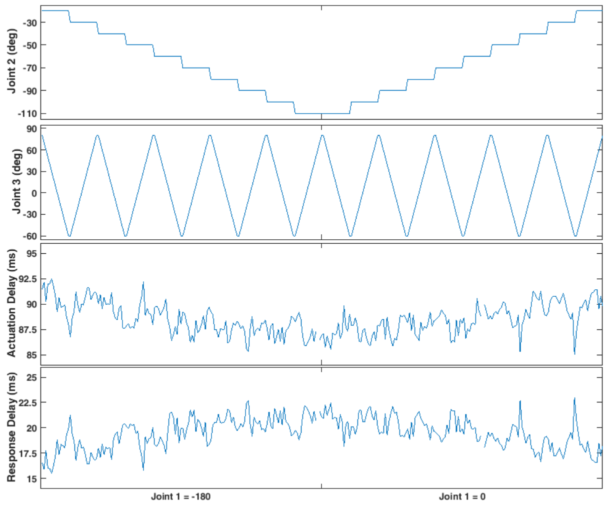

Figure 3.4: Delays of the Motoman 5-sixx robot's joint 1 moving in positive direction at two different positions, as a function of the positions of joint 2 and joint 3

then compared to the average error of just using the median delay of the robot and assuming the delay constant at that value. The resulting comparison of the actuation and response delay can be seen in Table 3.2 and 3.3, respectively.

As it can be seen, all the models generally does a good job of predicting the actuation delay. Especially the GPR, which outperforms the median's accuracy in all cases except for joint 2 on the UR. All the machine learning algorithms have problems with that joint, though. The boxplots in Figures D.8 and D.10 in the paper also shows that joint 2 performs somewhat different from the other joints. Except for joint 2, the machine learning algorithms shows the best improvement on the UR, compared to the median.

For the response delay there is also something to gain from using the learned models. No method seems better than other though.

GPR furthermore has the advantage of giving a measure of uncertainty

Table 3.2: Mean error in milliseconds of model fit for actuation delay. Results from [4]

Kuka	Joint 1	Joint 2	Joint 3	Joint 5	Combined	Average
Median	2.27	2.68	4.61	3.51	3.67	3.35
NN	1.79	2.55	4.74	3.37	3.21	3.13
RT	1.99	2.48	3.74	3.76	3.33	3.06
GPR	1.85	2.27	4.30	3.39	3.48	3.06
UR	Joint 1	Joint 2	Joint 3	Joint 5	Combined	Average
Median	6.18	4.64	6.08	2.59	5.33	4.96
NN	4.08	5.32	3.86	2.49	3.12	3.77
RT	4.63	5.41	4.28	2.72	3.42	4.09
GPR	5.01	4.89	3.68	2.47	3.36	3.88

Table 3.3: Mean Error in milliseconds of model fit for reaction delay. Results from [4]

Kuka	Joint 1	Joint 2	Joint 3	Joint 5	Combined	Average
Median	2.13	2.37	4.30	3.09	3.33	3.05
NN	1.70	2.32	4.68	2.22	2.36	2.65
RT	1.86	2.21	3.62	2.31	2.37	2.47
GPR	1.63	2.17	4.23	3.11	2.63	2.75
UR	Joint 1	Joint 2	Joint 3	Joint 4	Combined	Average
Median	4.82	2.00	4.08	4.90	5.09	4.18
NN	4.11	2.48	4.24	5.22	4.84	4.01
RT	4.66	2.44	4.47	5.84	5.33	4.35
GPR	3.88	2.44	3.75	5.20	5.05	3.82

along with the prediction of the delay. As it is known that the sum of the response delay and actuation delay is result of the sample time times an integer value, this knowledge can be used to get even better performance from the GPR than the other methods, as this knowledge is not used in training the data.

More specifically this knowledge can be used by predicting both the actuation and response delay at a given position, and then use the prediction with the lowest uncertainty to calculate the resulting delays. So if I want to find the response delay of joint 3 on the UR, the uncertainty will probably be lower on the actuation delay and thus it can be found by taking the actuation delay and subtract that from the nearest multiple of the UR's 8 ms sample time.

For both actuation and response delay, the mean error is smaller for the Kuka robot. This suggests that the delay is more consistent on the Kuka and thus that most is to gain from using machine learning with the UR.

Chapter 4

Conclusions

In this thesis, I have presented several different methods for doing better sensor based real-time control of robots, with an emphasis on solving the challenges identified as part of the "Real-time controlled robots for the meat industry"-project. The published methods can be used to improve both reliability and automation, which is key factors in a production environment where tens of thousands of individually shaped and sized products are being handled every day.

In paper A I have shown how an expert system can be used to increase the robustness of a robotic solution by monitoring task execution for errors and exceptions. I furthermore show that it is possible to make the robot reach to such events in a way that minimizes the error's impact on the robot's mission.

In the paper I showed how the expert system can handle several different exceptions related to mission execution on a mobile robot, and in section 3.1 I discussed how the results could be elaborated and used in other areas of robotics like industrial robots.

I have described methods for autonomous calibration and recalibration between a laser range scanner and an industrial robot in papers B and C. It is important to have such autonomous methods for sensor based robots in the food processing industry, where daily cleaning risk changing the pose of sensors.

In paper B I present a calibration target that can easily be used between a sensor and a conveyor due to its flat design. Furthermore, a method for doing a calibration using the target is presented along with showing how the worst-case accuracy of the calibration can be calculated based on the sensor and calibration target characteristics. While the accuracy is sufficient for planning in the cruise phase, it will probably not suffice for planning an actual grasp.

Thus, in paper C I show how the previous method can be improved by using individual laser rays and exploit their ability to detect the target's edges much more accurately.

In chapter 2 I have described why it is important to know the timing characteristics of a robot when doing sensor based real-time control of robots. With a general knowledge about the actuation delay, it is possible to predict whether a given robot is feasible for solving a specific task.

In a concrete example, I showed how careful design of a driver could make an otherwise infeasible UR robot feasible by evaluating the different commands for the robot and employing a threaded design for network instructions on the robot controller.

The process of doing a full measuring and modelling of the delay has been described in paper D. Here it was shown that it was possible to do a good approximation of the delay using machine learning and thus save time in an otherwise time consuming full mapping of the delay.

With a more detailed mapping of the delay from either a fully measured delay mapping or a learnt mapping, I described how this could be used to do Forward and Inverse temporal prediction as well as Current state correction in section 2.6. Such predictions can make sensor based real-time planning and control much more accurate and less time consuming to develop, as the robot does not need to be temporally calibrated to the robot cell setup each time the robot needs to perform a new task.

With these contributions, it is possible to do more robust, reliable, and autonomous sensor based real-time control of robots.

4.1 Perspectives

Ongoing research within the area of sensor based real-time control of robots is still needed as the topic is so broad and spans from topics in computer science like machine learning and network delays over electrical engineering with sensors and control theory to mechanical engineering with more optimal robots and end effectors. As the application areas of sensor based robots continuously grows, it is unlikely that there will ever be a point where there are no more interesting topics to be researched and areas where performance cannot be improved.

Further perspectives for the work described in this thesis, involves more work within the area of dynamically handling delays in robotics, as this is an area relatively untouched in research. Despite the fact that most people who has practical experience with robots know to compensate for delay in some degree, a thorough literature study has not shown any work related to the actual physical delays in robots, named actuation and response delay in this thesis.

Along the same lines is the threaded robot controller, which makes it possible to both send instructions at a rate equal to that of the sensor rather than being limited by the controller, as well as decrease the actuation delay. According to the fs100_motoman ROS package, the current Motoman driver exhibit an actuation delay of approximately 200 ms. It could thus be interesting to see if that driver, as well as drivers for other robots, could be improved by employing the same threaded control approach and wisely choosing the used instructions.

It could also be interesting to look more into sensor - robot calibration. Especially the case where the sensor is not within the reach of the robot, which is not an uncommon sight in industry.

The idea of establishing a worst-case error of a calibration could also be applied to other already existing calibration methods including those using other types of sensors. This would make it easier to compare the performance of the many different calibration methods.

Finally the idea of using an expert system for monitoring and exception handling could be further elaborated and is an ideal candidate for combining with the intense research effort being put into more user friendly ways of describing tasks for instructing robots and computers.

Paper A

Exception detection and handling in mission control for mobile robots [1]

Thomas Timm Andersen[1], Nils Axel Andersen[1] and Ole Ravn[1]

[1]Department of Electrical Engineering, Automation and Control Group, Technical University of Denmark, Richard Petersens Plads 326, 2800, Kgs. Lyngby, Denmark

Abstract:
This paper introduces a method for robust, rule-based mission control for mobile robots in a modular framework. Due to the modularity of the framework, it is possible to use both hierarchical control and reactive behavior seamlessly to find solutions to both planned and unplanned event in the mission execution.

A demonstration example for office navigation is presented along with considerations for rules that should ensure robust solving of missions curve.

Keywords: Expert systems, Autonomous mobile robots, Hierarchical systems, Robust performance, Robot Navigation, Robot programming

[1]Reproduced from T. T. Andersen, N. A. Andersen, and O. Ravn. “Exception detection and handling in mission control for mobile robots”. In: *Proceedings of 8th IFAC Symposium on Intelligent Autonomous Vehicles*. Vol. 8. IFAC Proceedings Volumes (IFAC-PapersOnline). Elsevier Science, 2013, pp. 187–192

A.1 Introduction

For decades, autonomous mobile robots have been expected to have a major impact on tomorrow's society, with journalist often proclaiming that the robots are coming to take over the world. Yet, the robots that have actually left the labs are neither of a number nor with a functionality anyway near of what could have been expected, let alone taking over anything.
One of the major showstoppers in preventing autonomous mobile robots from making it out of the lab is the robustness of one of the most elemental skills needed; mission control. This is particularly apparent in navigation - when the robot loses its heading, gets lost, or stuck, it is impossible for the robot to solve a mission that is related to moving to a specific position. While research in different approaches to navigation are progressing fast, the algorithms usually falls short when applied to a real robot in the chaotic human world outside of a lab's static or controlled environment.

In general, modern sensor technology have aided researches in coming up with some impressive methods, but it will never be possible to guarantee no missed detections nor any false positives detections. Other problems can also disrupt mission solving, from obstructed wheels to crashed software.

A.1.1 Previous work

A recent survey of fault diagnosis and fault tolerant control for wheeled mobile robots is given in [17]. Several methods are analysed to conclude on the most common problems, and some future trends within fault diagnostics and handling, including integrating models, control, and knowledge in a uniform framework, is presented.

[18] provides an experimental implementation of a hybrid (mixed discrete state/continuous state) controller for autonomous underwater vehicles, using rules for strategic mission control and hard real time for motion control at an

executive level. Others, like [19] have also shown that using an expert system for mission control can be advantageous.

The Mobotware framework introduced at IAV2010 [16] presents a modular, socket-based framework, where plugins can be used to extend the robots capabilities, both in real-time and non real-time control.

The knowledge gained from these contributions have been used to develop a solution based on a rule-based expert system, that are able to handle all parts of mission control for a mobile robot, and does so in a modular framework.

When analysing the sensor input and the perceived output of the individual modules, the situations where a navigation algorithm falls short can be detected and thus gives the opportunity to degrade to another method gracefully by handling the exception and thus saving the mission. When the system can detect exceptions and handle them, algorithms that are known to be effective, but error-prone in some situations, can be used with great benefits, as long as an opposing behaviour to the error-related situation can be defined.

The logic behind this is that while it is very difficult in an algorithm to detect if something is logically wrong with the input or result, usually even an untrained human can easily see if the robot is behaving less than optimal. By formulating the reasoning that humans do to detect this into some rules and using it in an expert system to do sanity checks on input, planned output, and the current state of the modules, all together at once, a much more robust behaviour can be obtained.

What exceptions to look out for, and appropriate handling of these, is highly application specific. This paper will therefore, after introducing the framework for using expert systems in mission control and sketching an example scenario, give some thoughts for considerations that should be applied when designing robust mission control. The paper will focus on navigation related problems, but the method and considerations described are applicable to all problems related to autonomous mobile robots.

A.2 System architecture

The Mobotware framework is used for simulation and controlling of robots, while Jess [20] is used as the expert system doing the exception detection and handling. As argued in [21], the use of a hierarchical control method like Mobotware and a reactive control method like Jess can yield robust, flexible, and generalizable navigation. The two is tied together using a Jess package named JessMW, which is introduced.

A.2.1 Mobotware

The Mobotware framework has three core modules:

- *Robot Hardware Daemon (RHD)* Flexible hardware abstraction layer for real-time critical sensors.
- *Mobile Robot Controller (MRC)* Real-time closed-loop control of robot motion and mission execution.
- *Automation Robot Servers (AURS)* Advanced framework for processing complex sensors and non real-time mission planning.

These modules allows for a two-dimensional decomposition: temporal and functional.

The temporal dimension divides Mobotware into a hard and a soft real-time constrained section, where RHD can handle the sensors and actuators requiring strict timing like wheel encoders and motors, making MRC able to run the robot at a desired speed for a given distance, while AURS handles the non timing-critical sensors like cameras and processes the data from these sensors to extract information about the surrounding environment.

The functional decomposision divides the framework in levels of increasing abstraction from the hardware abstraction layer in RHD, to reactive execution in MRC up to perception and planning with AURS.

The modular architecture is further strengthened by the use of plugins to implement both sensor interfaces and data processing algorithms, thus making it possible to use i.e. different methods for navigation, without altering anything else in the system.

All the core modules in Mobotware are connected through low latency TCP/IP connections, making it possible to easily exchange information both between the modules as well as with external processes, and distribute the computations across several computer platforms.

A.2.2 Jess

The Java expert system shell is an expert system implemented in Java that processes a CLIPS-like rule-based language. It can do both forward and backward chaining of rules, and using its RETE network it can handle several 100.000s of rules per second on a modern computer, while maintaining a huge fact base[20].

As Jess is implemented in Java, it has many object-oriented features including a direct interface to Java components. This makes it possible, via Java libraries, to connect to other processes on the computer and create shadow facts that represent the knowledge obtained from the other processes. This ability has made Jess widely popular in as diverse fields as mobile robotics[22], web services[23], fuzzy logic[24], and diagnostics and learning[25].

A.2.3 JessMW

In order to bridge the gap between Jess and Mobotware, the Jess package JessMW is made available. Using Java, the functionality of Jess is expanded by making it possible to communicate with MRC and AURS using the TCP/IP connections. Using these connections, the data concerning the surrounding environment perceived by AURS can be pushed to Jess and turned into shadow facts. Using rules, Jess can then compare the perceived information with any prior or learnt knowledge of the environment and thus do a sanity check on the information before the robot tries to act on it. Not only can this be used to prevent actions on detections known not to be possible, but it can also be used both for making the robot branch into a searching behaviour, if something that according to the internal state should be found but is not, and for challenging the robots beliefs about its current status, which might require the robot to go back in order to reassess the current situation of itself and the environment.

Likewise, information regarding the robot's status, like current motion instruction and odometry information, is fetched from MRC and turned into shadow facts by JessMW. It is also possible to get information from the time-critical sensors using this connection, and as Jess fetches this information via MRC and updates its own local knowledge base, it does not need to lock sensor values during search operations. This ensures that there are no risk of violating any real-time constrains. This is an issue sometimes observed when combining expert systems and real-time critical systems, and is part of the motivation for using a modular approach.
Via the connection to MRC, Jess can also send, stop and flush current motion instructions which make it possible not only to detect exceptions in mission execution, but also to handle them by imposing another solution to the current mission [26]. It can also monitor the execution of said motion instructions and detect i.e. if the low level security function in MRC has suspended motion due to a blocking obstacle in front of the robot.

Finally, JessMW can communicate with several Mobotware-enabled robots simultaneously, making it seamless to exchange information about the environment between robots and help each other to detect any exceptions or abnormalities.

An overview of the framework can be seen in Figure A.1.

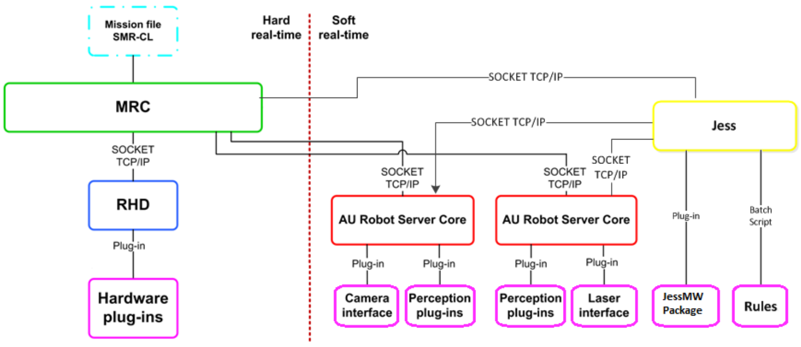

Figure A.1: Overview of the Jess-Mobotware framework

A.3 Implementation

To verify the proposed solution and the system architecture, extensive testing have been carried out, both in the methodological demonstration example described below as well as by several student at Automation and Control, DTU Electrical Engineering, who without prior exposure to robotics successfully used the principles to solve missions where cooperation between robots in accessing the state of the environment was a key element.

A.3.1 Demonstration scenario

The proposed solution was tested on a small mobile robot (SMR) running Mobotware. The SMR is a differential driven robot with wheel encoders for odometry, 1D IR distance sensor for low level collision prevention, laser scanner, and camera.

The robot was tasked to navigate through an office environment, which is controlled enough to ensure repeatability, but chaotic enough, due to human presence, to ensure problems for most navigation algorithms. Moving furniture in front of the robot or additional traffic can also easily be introduced to challenge the robot further. A partial map of the office space can be seen in Figure A.2

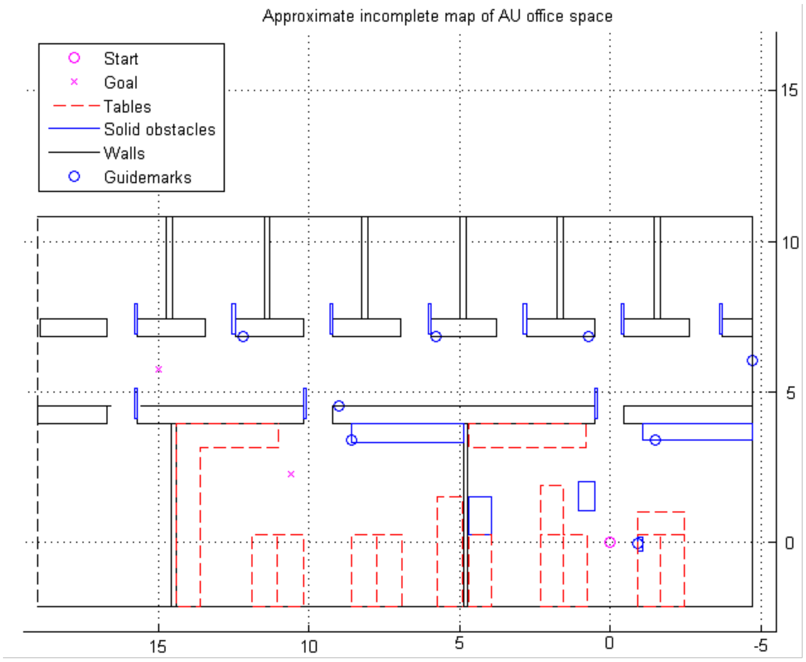

Figure A.2: Approximate map of office space

Even the optimal solution should require a travelled distance and enough turns to ensure the mission can't be solved robustly with odometry alone.

Also, two different goals was defined and specified at runtime, with multiple successive runs to each goal.

A.3.2 Algorithms for navigation

For global path planning, a graph node planner is used to calculate the most feasible path towards the goal. The expert system controls the path planner's nodes, making sure the knowledge of all landmarks' position is known by the path planner.

For global localization, landmark-based navigation is used. This approach has the advantage of only requiring a minimum of prior knowledge about the surrounding environment, namely locations of landmarks and traversable routes between then, compared to other global map-based methods.
It is implemented using 2D barcodes, easily detectable by the on-board camera. With proper calibration and optimal detection positioning, position and heading estimation relative to the barcode can be obtained to within ± 2 cm and $\pm 1^{\circ}$. As the barcode is printed on paper it is only detectable from one side.

For local path planning and obstacle detection, a method based on jumps and openings in a laser range scanner image is used[27]. It evaluates the environment in front of the robot and tries to move towards a given goal position. It is very opportunistic in that it plans a path and then executes it, without evaluating if something suddenly blocks the path. This is left to the low level collision avoidance to detect.

For local localization, odometry is used. To prevent the position error from growing unbounded, the odometry information is zeroed at each barcode detection and thus relating the local and global position.

A.4 Behaviours

To control the mission progression, at set of goal-seeking rules, or behaviours, are defined in the expert system. These can be divided into three groups: Input sanity validation, mission execution and exception handling. As stated earlier, the appropriate action for all the behaviours is a design choice, based on the specific application as well as the dependability of each sensor and algorithm.

A.4.1 Input sanity validation

This group of rules verifies all input related to mission progression. In the test example described in this paper, this is primarily related to the detection of landmarks.

If a landmark is reported spotted by the robot, the system should verify that the landmark is logically observable from the robots current position, i.e. that the robot is not currently behind the barcode, the relative orientation if off or the distance is further than what is known to be reliable for the detection algorithm. If the input is validated, the robot's position is updated based on this. But if the input is invalidated, the design choice of what action to take is based on what sensor output (odometry vs. landmark detection) is most reliable for the current robotic system.
One approach, and the one implemented in the demonstration, is to reject such detections, assuming they are reflections or simply false positives from the sensor.

Another issue is what to do if the system detects a landmark that it has no prior knowledge about. For a system operating in an assumed known environment, it might be most reasonable to ignore the detection, while a system designed for exploration probably would add the landmarks position, maybe flagging it unreliable until it is independently validated several times.

A.4.2 Mission execution

These rules govern mission execution in general, including planning, monitoring of progression, and issuing of motion instructions.

Using the path planner, the robot moves towards the goal position using the known landmarks en-route. Using odometry, the robot tries to position itself optimally accordingly to the barcode, so as to get the best estimate of position and heading. This is found to be at a small angle ($\sim 10^\circ$) relative to the barcode.

The locations visited, the current target location as well as planned locations to visit is all kept in the expert system's working memory.

A.4.3 Exception handling

These are the rules that have the ability to add real robustness to mission solving. The behaviours described above assumes some flow of events in a predictable order and that actions will lead to reactions, i.e. a motion instruction will lead to a corresponding movement or a landmark is visible and thus detected at a certain position.

But if this is not the case, behaviours should be defined that can detect the abnormalities, and then handle them in order to save the mission. This is also the major difference compared to input sanity validation, as an invalidated input reading should not jeopardise the mission, as long as a valid reading can be obtained afterwards without any active handling.

Obviously it is difficult for an algorithm, trying to detect something, to figure out whether it failed because nothing is there, or because something else is wrong.

This is where an expert system is critical, as it has the opportunity to diagnose or try redundant systems to detect what is wrong. But for the expert system to have any chance at this, it is necessary to have a broad understanding of what can lead to the used algorithms failing.

In the end, the appropriate action is highly dependent on the system's properties and the nature of the mission. If the detection algorithm is weak but localization is strong, it might be best to drive on, hoping to see the next landmark on the route. If both equally weak (or strong), it would probably be better to try and search for the landmark before giving up, and if it is the localization that is the weak part, maybe it would be better to go back the same route to a previous landmark and get a new position estimate to zero the local localization error. Or maybe a human operator should be notified, if the mission depends on the robot finding the landmark or it has been undetectable several times. In the test example, a searching behaviour is initiated.

The same goes for motion instructions; if something blocks the robot and thus prevents motion, a mission execution scheme that waits for the robot to finish its movement will stall if no thoughts are given to exception handling.

In the test example, an evasive behaviour is triggered if the low-level collision avoidance halts the robot, making it reverse and try to find another local path around the object. If that does not work, it will head back to the previous landmark to get a new orientation fix.

In general, when drawing a flow chart or state diagram of the expected mission progression, each transition should be analysed to map what part that might fail and thus prevent the transition. Likewise, each state should be analysed to consider what will happened if none of the expected transitions occur. Will the robot stall, will it keep moving in a possible wrong direction or could it might somehow end up in an undesirable state.

It is also worth to consider that the exception handling might also fail, and thus further degrading might be necessary. How redundant a system should be in the end is a design question that usually depends on a trade-off between desired robustness and resources, both time-wise, money-wise and computational power.

A.5 Results

In the presented test case, the robot successfully completed its mission in all 10 trials.

To show the robustness of the solution, different things were done to try and disrupt the mission control.

Figure A.3 is shown a map of a test run where the robot was placed at another orientation than what the internal state of the robot represented.

The red line represents the robot's own internal belief of where in the world it thinks it is, whereas the blue shows where the expert system's thinks the robot is, based on the sensed information regarding the environment. Thus the sudden jerks that only occur on the *worldpose* line happen when a landmark is detected.

This ability is similar to what other global localization methods can achieve, but goes to show that localization can be done in expert systems, and familiarizes the reader with the map structure used in the following. It should be noted that the overlaid map does not necessarily shows the true position of the robot, but is only added to give the reader a sense of where the robot is in the environment.

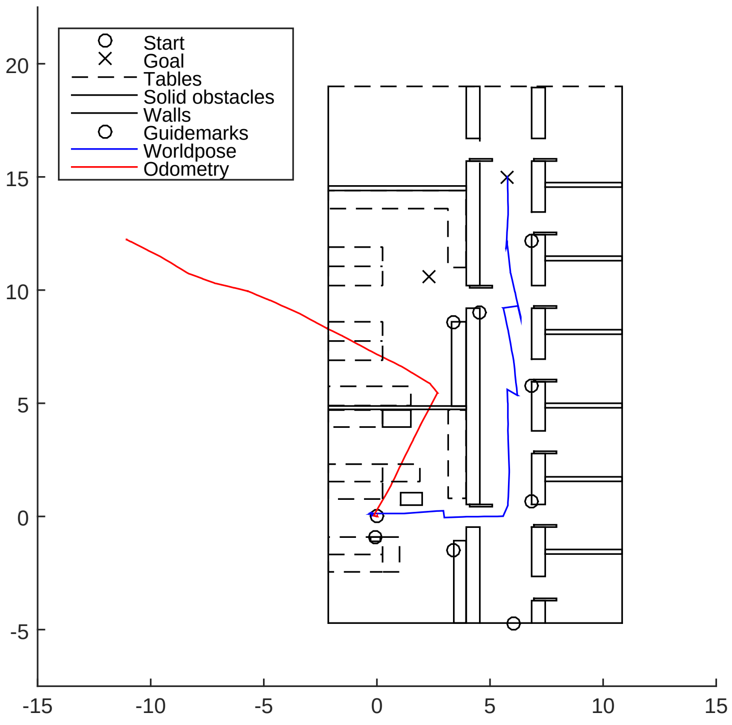

Figure A.3: Map plot of a demonstration test run

In Figure A.4 the robots path is suddenly blocked by an opening door. To avoid collision, the robot stops due to the low level collision avoidance. It then reverses and find a new path to the target.

The ability to detect and handle a security-related suspension of the execution was tested in several of the trials to prove that the robot could cope with a changing environment where humans move about and might unintentional place objects in front of the robot.

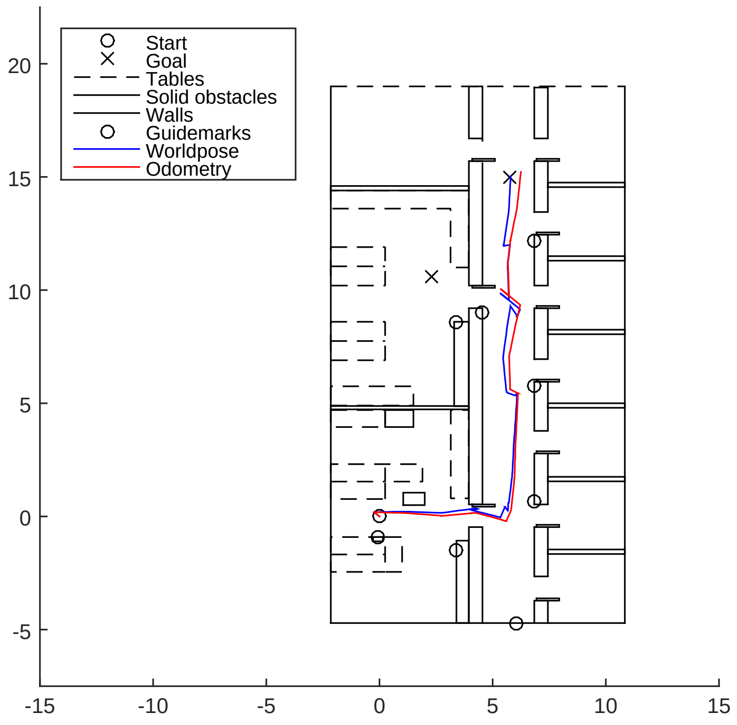

Figure A.4: Path suddenly blocked by an opening door

Figure A.5 shows the robot trying to find a landmark at an expected position. When this is unsuccessful, the robot starts searching for the landmark in a predefined pattern. This show an example of handling a sensor or detection algorithm that fails to deliver an expected result. This is a surprisingly common problem when moving a robot system from a simulated world to a real world implementation. The causes and consequences of such an event, or the opposite with a false positive detection, should be analysed with both great care and conservatism to increase the robustness of a mission controller, along with workarounds to the problem that works independent of the sensor or algorithm that is failing.

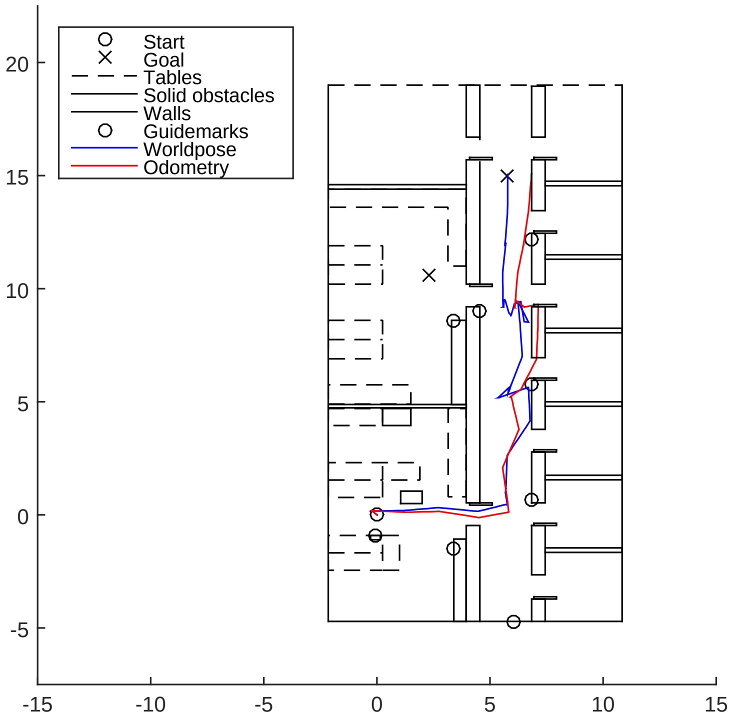

Figure A.5: The robot has to search for the landmarks

A.6 Conclusions

In this paper design for robust mission control using an expert system has been discussed. The field is highly application specific, so defining generic rules or a standard for designing robust mission control is not possible. Nevertheless, if the reliability of autonomous mobile robots is to gain enough trust to one day move out of the laboratories, this is a field where much improvement is needed.

It is shown that with appropriate care, a robust system can be designed and robustly solve navigational-dependant mission, regardless of what is literally thrown at it. The principle of using an expert system for robust mission control has also been proved by several inexperienced students in classes at Automation and Control, DTU Electrical Engineering.

Paper B

Calibration between a laser range scanner and an industrial robot manipulator [1]

Thomas Timm Andersen[1], Nils Axel Andersen[1] and Ole Ravn[1]

[1]Department of Electrical Engineering, Automation and Control Group, Technical University of Denmark, Richard Petersens Plads 326, 2800, Kgs. Lyngby, Denmark

Abstract:
In this paper we present a method for finding the transformation between a laser scanner and a robot manipulator. We present the design of a flat calibration target that can easily fit between a laser scanner and a conveyor belt, making the method easily implementable in a manufacturing line.

We prove that the method works by simulating a range of different orientations of the target, and performs an extensive numerical evaluation of the targets design parameters to establish the optimal values as well as the worst-case accuracy of the method.

[1]Reproduced from T. T. Andersen, N. A. Andersen, and O. Ravn. “Calibration between a laser range scanner and an industrial robot manipulator”. In: *2014 IEEE Symposium on Computational Intelligence in Control and Automation (CICA)*. IEEE. 2014, pp. 1–8

B.1 Introduction

Sensors have been used for adding perception to robotic manipulators in academia for years, and recently have also become more common in industry. To use the sensors, a proper transformation between the sensor and the manipulator is often necessary, so the position of the perceived object can be related to the position of the manipulator. For simple binary sensors like switches, the transformation, obtained through a calibration, will often only require information in one or two dimensions. This can usually be obtained with good measurements, timing or rigid construction. More complex sensors, like cameras, range measuring sensors, and moving sensors require both translational and rotational transformation for proper control. Especially for these complex sensors, this extrinsic calibration can be non-trivial and often requires data processing.

Robots working in food processing have an added need for quick and autonomous calibration, as daily cleaning of the entire production environment, using high-pressure hot water for cleaning and disinfection for example, can easily result in unintentional movement of a sensor. A small movement or rotation of a sensor will result in imprecise operation of the robot, which can result in outcomes from sub-optimal yields to destroyed products and equipment. It is therefore necessary to have a simple (re)calibration method that can easily be employed, even by non-technical staff.

B.1.1 Previous work

Much work has been done in the area of calibration for many types of sensors, including laser range scanners, which are the focus of this article. In [28], Antone and Friedman present a class of three-dimensional calibration targets that makes it possible to calculate the scanner's position relative to the calibration target using only a single scan. This leaves the task of determining the transformation from the target to the manipulator, which could be solved by attaching the target to the manipulator in a known configuration. The calibration targets are of a polypod type and should have a minimum of 4 legs, spanning a minimum area of about 1 m^2 for reliable calibration. Targets of this size can be difficult to fit between a laser scanner and an assembly line like a conveyor belt.

Adding space to move the calibration target around machinery and safe moving by the robot also wastes valuable manufacturing space, so this solution is not feasible in most industrial settings.

Hvilshøj et al. [29] use both a laser scanner and a camera for doing either high-speed or high-precision calibration of a mobile manipulator in different workspaces. The strategy of using a rigidly attached camera to determine the transformation between camera and laser scanner is generally well explored [30] [31] [32], but requires additional sensors, which in turn need additional calibration.

Pradeep et al. [33] use the intensity data from a tilting 3D laser scanner to obtain the transformation between the scanner and an end effector. This requires both a sensor that can detect intensity as well as either a 3D scanner or an actuated mount.

B.1.2 Contributions

In this paper, we present a robust method for obtaining homogeneous transformation between a laser range scanner and a robot manipulator by using a two-dimensional calibration target that can easily fit between a sensor and a conveyor belt. The initial calibration requires some human interaction to move the calibration target into the field of view of the sensor, while the method can be used for fully autonomous re-calibration.

The method is tested in simulation and the expected accuracy evaluated based on sensor resolution. This evaluation of worst-case precision is another significant contribution of this paper, as this is the first calibration method between laser scanners and robotic manipulators that includes a way for determining the worst-case accuracy.

Any type of 2D distance measuring sensor can be used, as there is no requirement to sample speed or meta-data such as intensity. It is assumed, though, that measuring noise is Gaussian and thus can be ignored by averaging over a sufficient number of samples, once static noise have been eliminated by calibration. This is a common assumption and is part of many sensor models like that of Thrun et al. [34]. The method can be used on any kind of robot, as long as it is possible to attach the calibration target to the robot, for example by using a gripper, and move the calibration target a known distance along a straight line.

By attaching the calibration target to the end effector, a transformation between the laser scanner and the end effector is found, which makes it possible to move the end effector precisely into the scanner frame without any inherent errors from imprecise forward kinematics.

B.1.3 Organization of the paper

In this paper we firstly present the fundamental mathematics used, and in section B.2 name and describe the relevant coordinate frames. In section B.3, we describe the process of finding the transformation, and how to design a proper calibration target. We describe our simulation and real world accuracy in section B.4 before detailing our conclusions in section B.5.

B.2 Fundamentals

In this section the fundamental mathematics used in this paper will be described. We will cover what is meant with a homogeneous transformation and how the frames are aligned.

B.2.1 Homogeneous transformation

According to Spong et al. [35], a homogeneous transformation is "a matrix representation of a rigid motion" (p. 61). It is used to describe the relative position and orientation of two coordinate frames. In this paper, we will only deal with frames in a 3 dimensional space.

The most general homogeneous transformation, [35] says, may be written as

$$H_1^0 = \begin{bmatrix} n_x & s_x & a_x & d_x \\ n_y & s_y & a_y & d_y \\ n_z & s_z & a_z & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} n & s & a & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.1}$$

In (B.1), $n = [n_x, n_y, n_z]^T$ is a vector representing the direction of the x-axis of *frame 1* in *frame 0*'s coordinates, $s = [s_x, s_y, s_z]^T$ represents the direction of the y_1-axis in *frame 0*, and $a = [a_x, a_y, a_z]^T$ represents the direction of the z_1-axis in *frame 0*. The vector $d = [d_x, d_y, d_z]^T$ represents the vector from the origin o_0 to the origin o_1 expressed in *frame 0*'s coordinates. Describing only the rotations can be done with the expression in (B.2)

$$R_1^0 = \begin{bmatrix} n_x & s_x & a_x \\ n_y & s_y & a_y \\ n_z & s_z & a_z \end{bmatrix} \tag{B.2}$$

B.2.2 Coordinate frames

In this paper we will use two frames; the robot's tool frame **T** and the laser scanner's frame **L**. In robot control, the robot's base frame **B** is often used as the ground truth or base reference frame and set to $[0,0,0]^T$. Using forward kinematics, the tool frame **T** can be determined based on the robot's Denavit-Hartenberg parameters and the joint values. Finding the homogeneous transformation between the laser scanner's frame and the robot's tool frame H_T^L is the purpose of this paper.

For robots having a gripper as the end effector, the origin of the tool frame o_T is commonly placed symmetrically between the fingers of the gripper, see Figure B.1. The z_T axis is pointing outwards from the gripper between the fingers in what is usually referred to as the *approach* direction. This is also the reason why column three in (B.1) is named a. Likewise, the sliding direction of the fingers is called s and is defined to be along the y_T-axis. The x_T is then normal to the plane formed by a_T and s_T, and is thus called n_T.

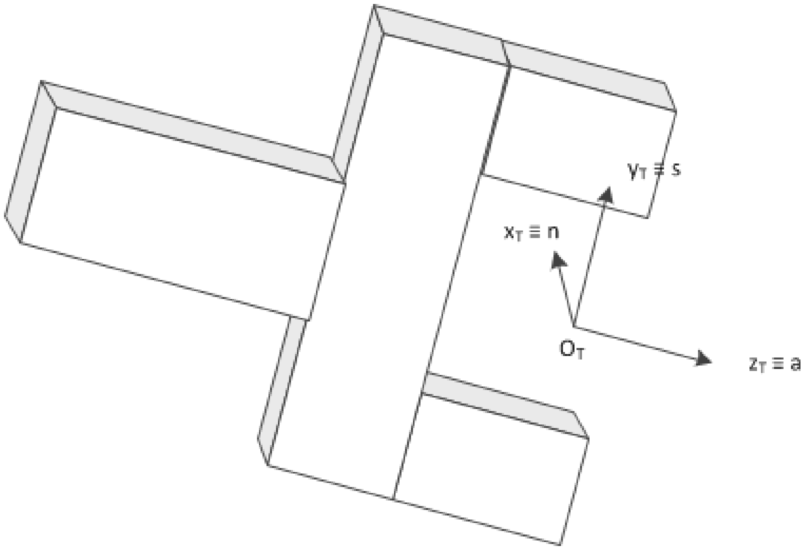

Figure B.1: Tool frame orientation

The laser scanner makes 2D range data in a plane, based on laser rays emitted in discrete angles, denoted by θ_i. For each ray, a distance r_i is measured along that ray. Following convention, and without loss of generality, we place the origin of frame **L** at the origin of all laser rays emitted from the laser scanner, with the z_L-axis normal to all laser rays emitted by the scanner, the x_L-axis coincident with the ray at $\theta_i = 0$, and the y_L-axis coincident with the ray at $\theta_i = \pi/2$, forming a right-handed frame. This is shown in Figure B.2.

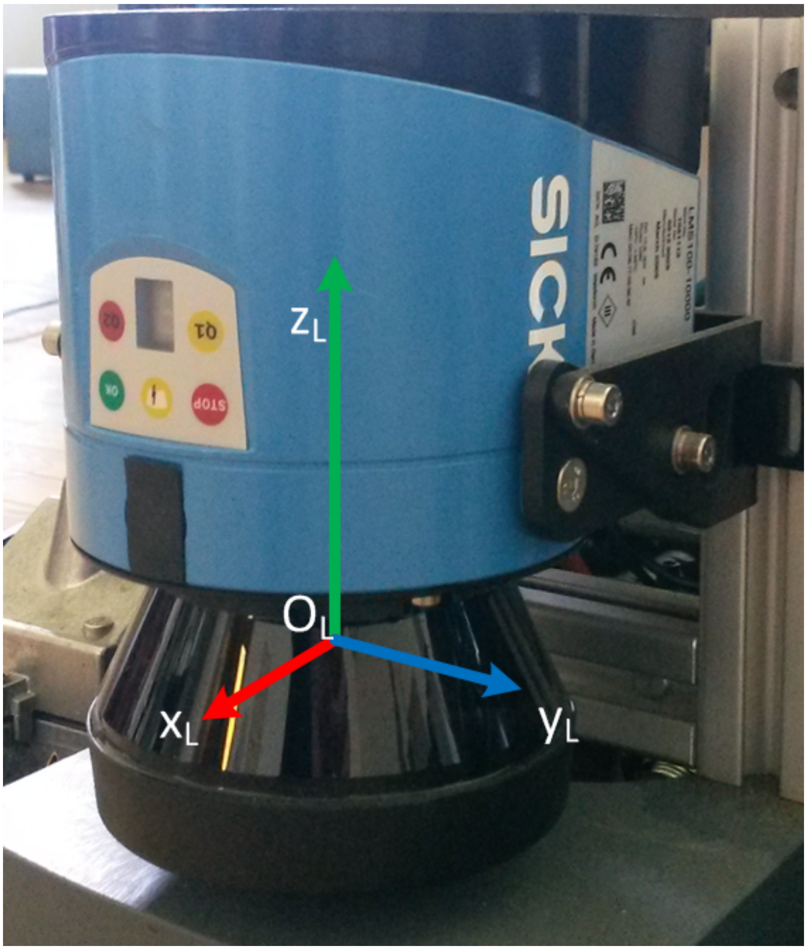

Figure B.2: Laser frame orientation

B.3 Methodology

In this section we describe the method for finding the four vectors given in (B.1).

In designing a simple calibration target, it is relevant to see what can be learned from a simple geometric shape like a square with a known width of w. Without loss of generality, it is assumed that the calibration target is rigidly attached to the end effector and placed with the surface towards the laser scanner. The surface of the target should be in the plane spanned by z_T and x_T, and the end effector should be moved so z_L and z_T are as near to parallel as possible. This is shown in Figure B.3.

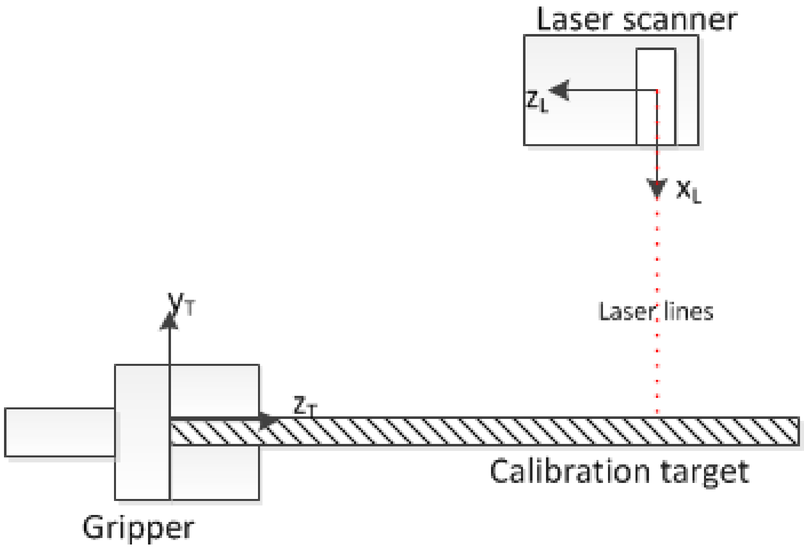

Figure B.3: Laser frame orientation

It is assumed that the laser measurements are filtered (i.e. by using a distance threshold) and added to the set M_k, so only the laser ray measurements hitting the calibration target is added, and sub-sequentially converted to points in Cartesian space in the laser scanner frame **L**. The subscript k in M_k indicates the scan number. Two scans are required, with the calibration target moved a known distance d in the y_T direction between the two scans.

B.3.1 Finding the rotations

To find the rotation of the calibration target relative to the x_L-axis, we exploit our knowledge of the width of the calibration target w relative to the width measured. Using the two most distant points in either M_1 or M_2, we find the distance between those point.

$$w_{Meas} = \sqrt{(p_1.x - p_2.x)^2 + (p_1.y - p_2.y)^2} \tag{B.3}$$

Using the Law of sines, we can then determine the rotation angle

$$Rot_{x_L} = \frac{\pi}{2} - \arcsin\left(\frac{w}{w_{Meas}}\right) \tag{B.4}$$

It is not possible to know whether it is a positive or negative rotation. Placing the target at a slight angle that we approximately know, we can define the sign of the rotation.

Using the same two points, the rotation around the z_L-axis can be found as the slope of a line given by those two points. The rotation angle is

$$Rot_{z_L} = \arctan\left(\frac{p_2.y - p_1.y}{p_2.x - p_1.x}\right) + \frac{\pi}{2} \tag{B.5}$$

To find the last rotation around y_L, we need to know the distance the calibration target has moved between scan M_1 and M_2 in the z_L and y_L direction as a result of the motion along the y_T axis. We could use the same approach as above with measuring the change in the y_L direction and relate it to the expected change if the rotation is 0. This method relies on the equation in (B.6).

$$Rot_{y_L} = \frac{\pi}{2} - \arcsin\left(\frac{d}{d_{meas}}\right) \tag{B.6}$$

In a space constrained production environment, horizontal movement of more than 15 cm might be difficult to do, and a worker should be able to position the calibration target closer to horizontal than 10°. This would result in a measured maximum displacement of 15.23 cm, or a difference of 2.3 mm. This is less than the statistical error of many laser scanners. So while it would work in theory, it will not yield a good result in practice.

To get a better result in practice, we need to be able to measure the length of the last side of the triangle, we call it a. This is, however, not possible to measure from the scans using the simple square calibration target. Below we will present a calibration target from which this distance can also be measured with a laser scanner.

B.3.2 Designing a calibration target

Based on our analysis so far, we have three design criteria for our calibration target

- Known width: An easily distinguishable section of the calibration target must have a known width to find the x_L and y_L rotations.
- Varying feature: To measure the translation or the target, a part of the calibration target must have a varying feature that makes it possible to determine either where on the target the scans were made, or how far the target has moved.
- Small footprint: For ease of use in an industrial setting, the target should be as small and flat as possible.

To satisfy the first requirement, the main part of the target should still be a square or rectangle.

For the varying part, this could either be a varying height of some of the target, or a varying width parallel to the known rectangle. To support the last requirement in terms of keeping the target flat, some of the target width is

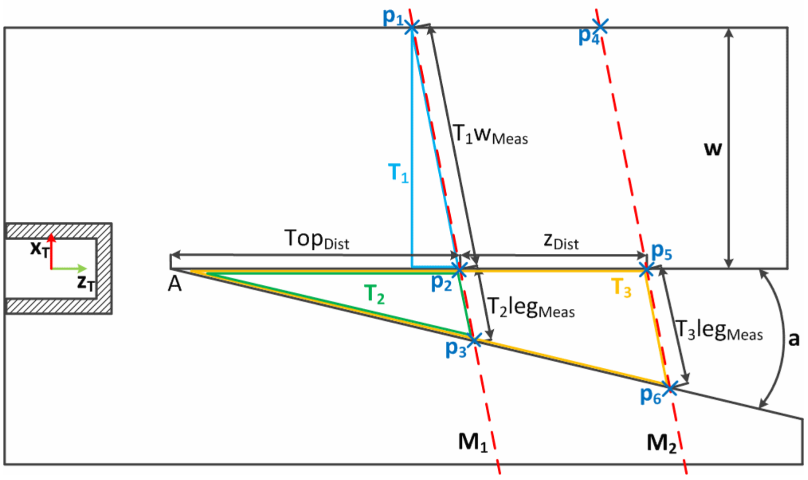

Figure B.4: Calibration target. Two laser scans are shown in red, and the three triangles of interest are drawn in blue, green and yellow

made varying.

A sketch of a calibration target that support this can be seen on Figure B.4. The grayed-out "shoe" on the left is for the gripper's finger; this ensures that the gripper holds the target in a known pose, thus ensuring that the transformation from gripper to target is known. It is assumed that the shoe is designed so that geometrical errors in the grasp in negligible. This could also be ensured by casting the target with a mechanical coupling that attaches firmly to the manipulator. The only requirement is that the transformation from the manipulator's base link to the target's frame is known.
A setup showing the calibration target held underneath a laser scanner by a robot can be seen in Figure B.5.

A part of the figure is parallel to the shoe and has one edge aligned with the center-line of the shoe. This part of the figure has a known width of w as shown on Figure B.4 and is part of the blue triangle T_1. Knowing the two sides w and $T_1 w_{Meas}$ of the right-angled triangle makes it possible to determine all the angles of the triangle and thus the rotation around Rot_{x_L} as shown in (B.4).

The distance between the two "legs" are varying, and by measuring that distance (annotated as leg_{Meas} on Figure B.4), the distance to the top of the green triangle T_2 and the yellow triangle T_3 can be found based on the Law of sines and the knowledge of the angles A and Rot_{x_L}. For the green triangle T_2, this distance can be found using (B.7) and (B.8), while the top_{Dist} for the yellow triangle T_3 can be found using points p_5 and p_6 in (B.7).

$$leg_{Meas} = \sqrt{(p_2.x - p_3.x)^2 + (p_2.y - p_3.y)^2} \tag{B.7}$$

$$top_{Dist} = \frac{\sin(\frac{\pi}{2} - Rot_{x_L} - a) \cdot leg_{Meas}}{\sin(A)} \tag{B.8}$$

The difference between the two top_{Dist} equals the unknown side length a of the triangle needed to find Rot_{y_L}. Knowing this, along with the distance traveled in the y_T direction, the last angle Rot_{y_L} can be found as shown in (B.10). In (B.9) we denote the calculated length d_{meas} to show that this is the same value we measured and used in (B.6) earlier, although it is now calculated.

$$d_{meas} = \sqrt{(T_3 top_{Dist} - T_2 top_{Dist})^2 + d^2} \tag{B.9}$$

$$Rot_{y_L} = \arcsin\left(\frac{T_3 top_{Dist} - T_2 top_{Dist}}{d_{meas}}\right) \tag{B.10}$$

B.3.3 Finding the transformation

Having found the rotations of the calibration target relative to the laser scanner, writing the transformation is now trivial. Using the shorthand notation $c_x = cos(Rot_{x_L})$, $s_y = sin(Rot_{y_L})$, $s_{xy} = s_x \cdot s_y$ etc., the rotation between the

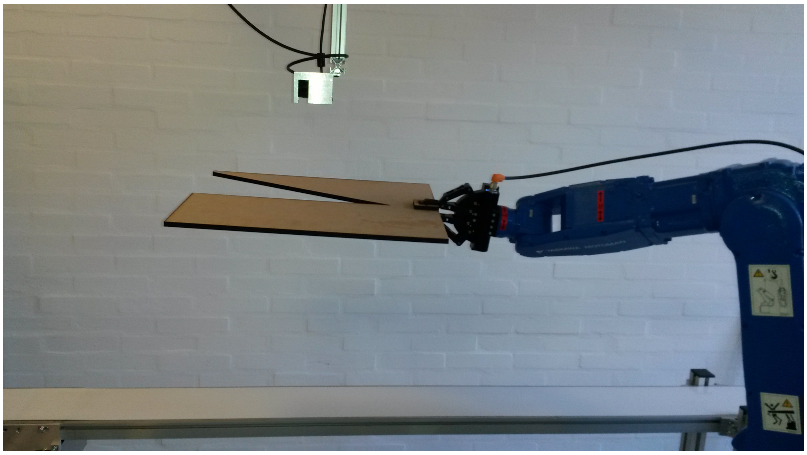
Figure B.5: Concept photo showing a robot holding a calibration target underneath a laser scanner

laser and tool frame is given in (B.11)

$$R_T^L = \begin{bmatrix} c_{yz} & -c_y s_z & s_y \\ c_z \cdot s_{xy} + c_x \cdot s_z & -s_{xyz} + c_{xz} & -s_x \cdot c_y \\ -c_{xz} \cdot s_y + s_{xz} & c_x \cdot s_{yz} + c_z \cdot s_x & c_{xy} \end{bmatrix} \tag{B.11}$$

The translational part when the gripper is at the position of the M_2 scan is given by the expression in (B.12), where $A_{Dist} = T_3 top_{Dist}$ + the distance from the calibration targets coordinate frame center to the top angle point A of the target.

$$d_T^L = R_T^L \begin{bmatrix} p_5.x \\ p_5.y \\ A_{Dist} \end{bmatrix} \tag{B.12}$$

The transformation matrix between the laser and the gripper will change whenever the gripper is moved, as reflected in (B.12). Multiplying the transformation with the transformation between the gripper and the robot base H_B^T would give the constant transformation between the laser scanner and the robot's base H_B^L.

Instead of moving the target in the y_T direction, one might intuitively consider moving the target in the z_T direction to get a 3D point cloud of the target, and then employ techniques such as plane detection, edge detection, or model fitting to find the orientation of the target. But by examining Figure B.3, it is clear that motion in the plane spanned by z_T and x_T won't affect the measured distance in the x_L direction, no matter how the target is rotated with respect to y_L. This rotation would thus be lost, and therefor motion in the y_t direction is a necessity. Thus two complete surface scans would be needed.
As the error in the scanner is assumed Gaussian, taking many measurements at one pose, average and then use the method above would yield the same, or better, results as creating a point cloud and then use detection techniques, as the target unavoidably will vibrate a little while being moved, and matching a measurement to a specific pose could also introduce errors. These models won't add anything to the accuracy of the position of the edges that averaging over a stationary target won't give. Such a process will also add additional time consumption where the robot won't be producing anything.

B.4 Evaluation

For evaluating the proposed method, we will first do a simulation of the method to show that it outputs correct transformations. Next we will do a numerical evaluation of the calibration target's design parameters, to see what accuracy can be expected with the proposed method and a laser range scanner.

B.4.1 Simulation

To verify the method, we have made a script that draws the calibration target, rotates it around its axes and calculates the position of p_1 to p_6. Using this, we can verify that the method works for all combinations of rotations.

Looping over a range of combinations of rotations, from $+25°$ to $-25°$, we verified that the method worked within that range. That range was selected as we expect a human operator easily can place the target within that range. A small random value was added to each angle at each iteration to catch any systematic errors. The method was shown to work with all combinations of angles within that range. Output from different simulations can be seen on Figure B.6 and B.7.

On the first three figures, the rotations are only around one axis, while the target is rotated around all three axes on the last plot. For rotations not including a rotation around the y_L axis, the calibration target can only be seen in one position, as the targets are overlaid. It can be difficult to see a rotation on the plot showing rotation around the z_L axis, but the fixed width part only looks 18.3 cm wide, where it is actually 20 cm wide, revealing a slight rotation.

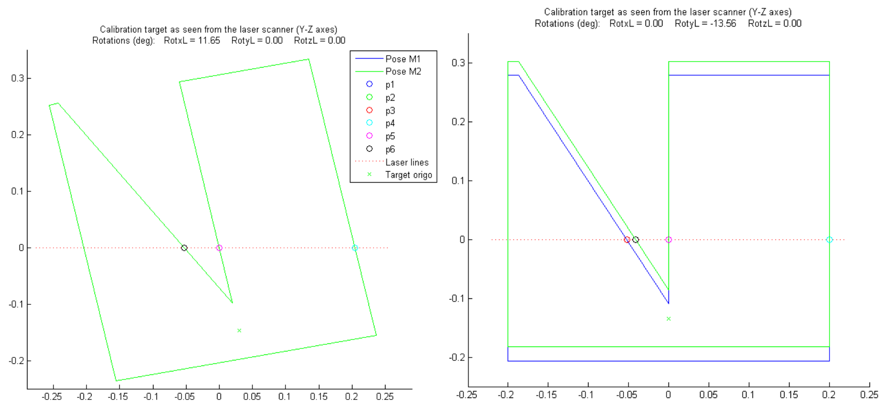

Figure B.6: Simulated rotations as seen from the laser scanner. Left: Rotation around x_L. Right: Rotation around y_L

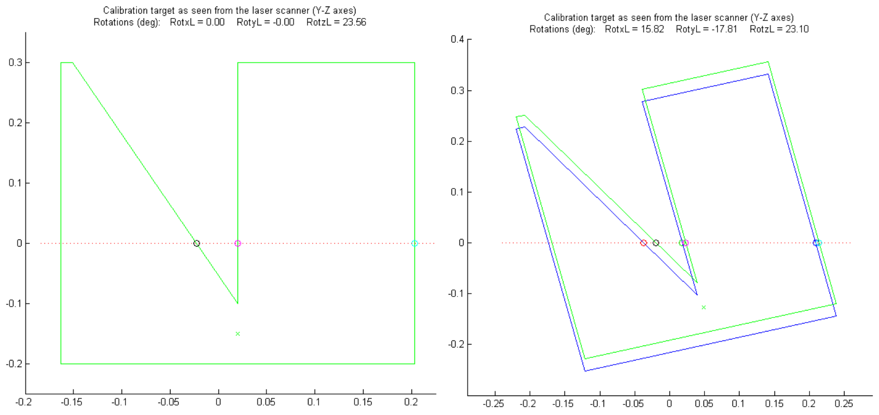

Figure B.7: Simulated rotations as seen from the laser scanner. Left: Rotation around z_L. Right: Rotation around x_L, y_L and z_L

B.4.2 Numerical design evaluation

In this section we will evaluate the design parameters of the calibration target to see how they affect the accuracy of the proposed method. The parameters in play are the width of the rectangular part of the target w and the angle of the triangle top point A. Also the distance from the sensor to the calibration target $dist_{Target}$ influences the accuracy.

Most laser scanners, like the Hokuyo URG-04LX[36], have a length measurements resolution of 1 mm with an accuracy of ± 1 cm. The angular resolution varies from product to product; most of Hokuyo's scanners have a resolution of $0.35°$, while Sick offers resolution from $0.5°$ to $0.125°$. No variation is given for the angular reading, so this is assumed to be negligible. As this resolution is given in degrees, and for ease of reading, the rest of the section will use degrees for measuring angles. Results of trigonometric functions are converted to degrees as well.

The largest source of measurement errors is quantization errors due to the angular resolution, and noise of the length measurement. Hokuyo does not state how much of the ± 1 cm variation originates from systematic error and how much from statistical, but our measurement shows that 50 % of the measurements have an error of $\sim\pm 0.3$ cm or less. This is similar to other research like the study of the same sensor made by Kneip et al.[37]. By calibrating the sensor to the material of the calibration target, better performance is attainable. As the error is Gaussian, we can average over multiple samples and/or fit a line to the data points and further minimize the error of the length measurements. The statistical error of Sick's scanners are in the range of ± 1.2 to ± 0.3 cm, and is expected to be minimizeable using the same techniques as with the Hokuyo.

The main error source is thus the angular quantization, which shows as a fixed angle between each laser beam. Thus the further away from the scanner the target gets, the wider the space between the laser beams is. This suggest that the target should be as close to the scanner as possible. It would therefor make sense to use the scan closest to the scanner (points p_4 and p_5) for finding the rotations around x_L and z_L. The target should also be as narrow as possible, since a wide target will have its edges further away from the scanner. A 3 cm wide target held 5 cm from the Hokuyo scanner will be measured to be 2.998 cm wide (an error of 0.05 %), whereas a 30 cm wide target at the same distance will be measured to be 29.901 cm wide (an error of 0.33 %).

We cannot make the target too small, as we also need some points for the line fitting. For our analysis we will use the values $w = 0.05, A = 40^\circ, T_2top_{Dist} = 0.02, d = 0.1, dist_{Target} = 0.05$, where $dist_{Target}$ is the smallest distance from the laser scanner to the calibration target during the second scan M_2. All the distances are in meters. This ensures that roughly 50 rays will hit the fixed-width part of the target even at scan M_1, although the exact number depends on the rotation around x_T.

If the target is structural strong enough, the leg with the varying width can be made arbitrary wide to give more points for the line fitting.

With the center-line of the calibration target directly below the laser scanner ($p_5.y = 0$), we can determine how many laser rays that will hit the calibration target and thus the accuracy of the measured width. For the Hokuyo the measurement will be

$$\begin{aligned}
dist2p_4 &= \sqrt{w^2 + dist_{Target}^2} = \sqrt{0.05^2 + 0.05^2} = 0.07 \\
angSpan &= \arcsin\left(\frac{w}{dist2p_4}\right) = \arcsin\left(\frac{0.05}{0.16}\right) = 45^\circ \\
nrRay &= floor\left(\frac{angSpan}{angResul}\right) = floor\left(\frac{45}{0.35}\right) = 129 \\
w_{meas} &= \frac{\sin(nrRay \cdot angResul) \cdot dist_{Target}}{\sin(90 - nrRay \cdot angResul)} \\
&= \frac{\sin(129 \cdot 0.35) \cdot 0.05}{\sin(90 - 129 \cdot 0.35)} = 0.04996\,\mathrm{m} \qquad \text{(B.13)}
\end{aligned}$$

Variations of the target's width between 4.996 cm and 5.058 cm cannot be measured, meaning rotations around x_L of up to

$$\begin{aligned}
err_{rot_{x_{max}}} &= 90 - \arcsin\left(\frac{w}{w + w_{meas_{max}} - w_{meas_{min}}}\right) \\
&= 90 - \arcsin\left(\frac{0.05}{0.0506}\right) = \pm 8.95^\circ \qquad \text{(B.14)}
\end{aligned}$$

will go unnoticed. Placing the target more underneath the scanner, ideally so $p_4.y = -p_5.y$, will decrease the worst-case error to $\pm 7.07^\circ$. It is not possible to predict the effect of varying w or $dist_{Target}$, as the error will be zero whenever a laser beam exactly hits both edges of the target. It is not possible either to predict the effect of rotation, as this has the same effect as changing w. It should be noted that these are worst-case errors, and that using another scanner with better angular resolution would also help significantly. Using SICK's LMS400 scanner with an angular resolution of 0.125° will yield a maximum error of

$\pm 4.23°$.

Due to noise, both the $p_4.x$ and $p_5.x$ length measurement can be up to ± 1 cm off, resulting in an error of up to

$$\begin{aligned} err_{rot_{zmax}} =& 90 - \arctan\left(\frac{w}{2 \cdot err_{meas}}\right) \\ =& 90 - \arctan\left(\frac{0.05}{0.02}\right) = \pm 21.8° \end{aligned} \tag{B.15}$$

This error is solely due to noise in the length measurements, and as stated above, these can be minimized significantly with calibration, averaging and line fitting. With our measured error of $\sim\pm 0.3$ cm or less, the maximum error will be decreased to $\pm 6.8°$. Rotation around z_L will decrease the error relative to the actual difference, effectively increasing the accuracy marginally. By fitting a line to the $nrRay = 129$ data points across the calibration target, the error is likely to be decreased substantially. Increasing w would give more data points for this line fitting and thus increase the accuracy, but as stated above could negatively affect the width measurement and thus $err_{x_{max}}$.

The distance between p_5 and p_6 will according to (B.8) be $leg_{Meas} = 1.68$ cm. If the center-line of the calibration target is once again placed directly under the laser scanner (so $p_5.y = 0$), variations in this width between 1.65 cm and 1.68 cm will go unnoticed with the Hokuyo. At the other placement suggested above ($p_4.y = -p_5.y$), the variations would be between 1.67 cm and 1.70 cm. Inserting the variations in (B.9) yields a variation on Top_{Dist} of less than ± 0.1 cm and using (B.10), the maximum unnoticed angle will be

$$err_{rot_{ymax]}} = \arcsin\left(\frac{T_3Top_{Dist} - T_2Top_{Dist}}{d_{meas}}\right) = \begin{cases} +2.64° \\ -0.53° \end{cases} \tag{B.16}$$

Increasing d will increase the accuracy, just as placing the hole underneath the laser scanner will (so $p_2.y = -p_3.y$). These changes could have a negative effect on the other readings though.

The translational part of the transformation matrix is affected by how accurate we can find p_5.

Based on the above analysis, we should be able to get a result with a linear error of ± 0.1 cm at most and no angular errors of more than 4.22° with the best available scanner - and this error could be made smaller by proper placement of the target under the laser scanner. The error is smallest closest to the plane spanned by x_L and z_L, suggesting that the optimal position is when $p_1.y = -p_3.y$. Placing the target in just the right place, so a laser beam hits exactly all 6 points will result in no error on rot_x or rot_y. By carefully moving the robot in tiny increments, it should be possible to get the target into the correct position. Once a human operator have done an initial calibration within $\pm 5^\circ$, the robot can do this autonomously.

B.5 Conclusions

In this paper we have presented a method for finding the transformation between a laser scanner and an industrial robot manipulator. Unlike previous work in the field, like that of Antone and Friedman [28], Hvilshøj et al [29] and Pradeep et al.[33], our method does not rely on other sensors, intensity readings or large calibration targets. By careful design, we have created a method that only uses a flat target, making it possible to fit in most production environment between a conveyor belt and the laser scanner.
We have argued why this method yields the same or better results than those based on building an entire 3D point cloud of the target and then employing plane detection, edge detection, or model fitting, both with respect to accuracy and especially time.

We have shown in simulation that the method can accurately detect rotations and translations around any and all axes, only limited by the measuring

accuracy of the laser scanner.
To evaluate the obtainable accuracy with different laser scanners, and show how the design parameters of the calibration target affects this accuracy, we have done a numerical design evaluation. It shows that even with a cheap laser scanner like the Hokuyo URG-04LX, the obtainable accuracy is below $\pm 10^\circ$ on all axes. While this accuracy is certainly to low for some applications, it is more than adequate in others, i.e. where the purpose of the cheap laser scanner only is to find the approximate pose of an object, before using other techniques like eye-in-hand, tactile, or force sensors to do the actual manipulation of the object.

Paper C

Optimizing the autonomous self-calibration between a robot and a distance sensor [1]

Thomas Timm Andersen[1], Nils Axel Andersen[1] and Ole Ravn[1]

[1]Department of Electrical Engineering, Automation and Control Group, Technical University of Denmark, Richard Petersens Plads 326, 2800, Kgs. Lyngby, Denmark

Abstract:
Robust and accurate calibration and re-calibration between sensors and robots are essential for flexible handling and manufacturing of products with natural variance. For documenting consistency of the end product, it is furthermore important to be able to determine the accuracy of the calibration.

In this paper, we take a previously published method for calibration between a laser range scanner and a robot arm and show how this method can be substantially improved by relying on the precise nature of an industrial manipulator to annul the limiting effects of the laser scanner's angular quantization. Furthermore, we show how the worst case accuracy can be calculated, and show how the method presented in this paper improves the worst case accuracy with a factor 166 for the translational calibration and between a factor 32 and almost 4.000 for the rotational calibration.

[1]Reproduced from T. T. Andersen, N. A. Andersen, and O. Ravn. “Optimizing the autonomous self-calibration between a robot and a distance sensor”. In: *Robotics and Computer-Integrated Manufacturing* (2016)

C.1 Introduction

For robots to engage in dynamic and changing environments, they need sensors to reason about their surroundings. These sensors, in turn, needs to be properly calibrated relative to the robot in order for the robot to relate to the perceived information. To this end, several methods exist to approximate the transformation between a robot and one or more sensors.

The complexity and required accuracy of these methods depends among other things on the sensor and its usage. For simple binary sensors like optical switches, a single offset measurement might suffice, whereas more complex sensors with multidimensional measuring capabilities might require extrinsic calibration of translational, rotational and temporal nature.

As the trend in modern manufacturing is moving towards flexible and sensory-assisted automation, the need for calibration and recalibration is even more distinct, as small misalignment can result in suboptimal or even destroyed products and damaged machinery. Having these procedures automated can further help to optimize the plant's uptime.

C.1.1 Previous Work

As a result of this, a lot of research have been done regarding calibration between robot arms and different sensors, including 2D laser range scanners which is the focus of this paper. Hvilshøj et al. [29] uses both a camera and a laser scanner for doing either high-speed or high-precision calibration of a mobile manipulator moving between different workspaces. The strategy of using a rigidly attached camera to determine the transformation between camera and laser scanner is generally well explored [31] [32] [30], but requires additional sensors, which in turn need additional calibration.

Pradeep et al. [33] use the intensity data from a tilting 3D laser scanner to obtain the transformation between the scanner and an end effector. This requires both a sensor that can detect both distance and intensity as well as either a 3D scanner or an actuated mount.

Antone and Friedman[28] suggests using a class of three-dimensional calibration targets that makes it possible to calculate the scanner's position relative to the calibration target using only a single scan. This leaves the task of determining the transformation from the target to the manipulator. The calibration targets are of a polypod type and should have a minimum of 4 legs, spanning a minimum area of about 1 m^2 for reliable calibration. Targets of this size can be difficult to fit in a confined production space between a laser scanner, a robot manipulator, and an assembly line like a conveyor belt. Adding floor space to move the calibration target to and from the robot and between machinery also wastes valuable manufacturing space, so this solution is not feasible in most industrial settings.

Another drawback of these and other existing methods is that they provide no metric for the accuracy of the method. To that end, we have previously published a method[2], which is both able to perform the calibration as well as provide a worst case accuracy of the calibration.

As the method we present in this paper builds on our previous work, as well as for completeness, the principal idea from [2] is described below. It relies on the geometrical relationship between the measured edges of a two fingered calibration target and a laser range scanner, which is used to find these edges. The calibration target, with overlaid geometric features, can be seen on Figure C.1. The grayed out shoe on the left is for rigid alignment of the gripper to the robot to ensure a known transformation between the robot and the calibration target. The coordinate system is thus shared between the gripper and the target and this coordinate frame is termed T. The distance w and the angle a are known design perimeters.

The method works by first taking a distance measurement and then moving the target in the Y_T axis direction, before doing another measurement. On Figure C.1 the two scans are shown as M1 and M2, with the range measurement series along the red lines. Using these measurements the points p_1 to p_6 can be located, which in turn can be used to determine the target's rotation

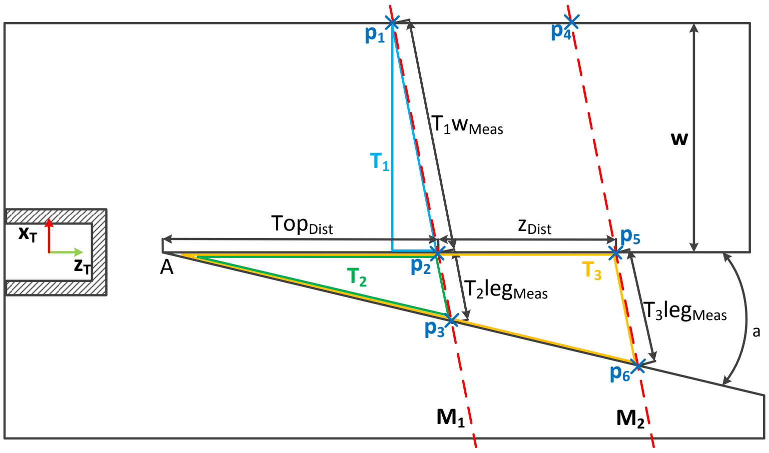

Figure C.1: Calibration target with overlaid geometric features. Reprint from [2]

relative to the scanner together with the law of sines and law of cosines.

If the target is rotated around the X_T axis relative to the scanner, the two scan lines won't be coincident. Based on the difference between the green triangle T_2 and the yellow triangle T_3 and the knowledge of how far in the Y_T direction the target was moved, the rotation can be found. Similarly the blue triangle T_1 can be used to find the rotation around the Y_T axis. The distance between p_1 and p_2 is the hypotenuse $T_1 w_{meas}$ and the adjacent is the known length w. The angle between these two lines are equal to the rotation around Y_T. Finally, the rotation around Z_T can be found from the distance between the scanner and the points p_1 and p_2.

The transformation from robot base to the target, and thus every point on the target is known a priori. As the position of the points p_1 to p_6 is also known in the laser scanner frame L, the translational transformation can routinely be found as well.

We also showed that the main source of the calibration error comes from determining the position of the calibration target's edges. This is due to the quantization of the laser scanner angles and can lead to errors as big as $\pm 10^{\circ}$ on all the rotations.

C.1.2 Contribution

This paper presents an optimal method for autonomous translational and rotational calibration between a robot and a distance sensor. The optimality is achieved by annulling the quantization error, using only one or two laser rays and motion of the calibration target to align the edges of the target with the laser rays, thus greatly improving the performance of the previous method.

In our research we have focused on calibration between an industrial manipulator and a 2D laser range scanner and that is also the focus of this paper, but the method is applicable to all robots with 6 degrees of freedom and both 2D and 3D distance sensors. As it will become apparent later though, the achievable accuracy of the calibration depends on the robot's ability to move straight in Cartesian space.

C.2 Methodology

The purpose of calibrating a sensor to a manipulator is determining the homogeneous transformation between the coordinate frames of the sensor and the manipulator. The coordinate frames between which we will determine the homogeneous transformation in this paper is the laser scanner's frame **L** and the robot's tool frame **T**. This can be seen on Figure C.2.

C.2.1 Design of calibration target

The two finger calibration target we use in this paper is identical to that of [2] as the analysis that lead to the design still holds true. The target is drawn on Figure C.1 with black lines.

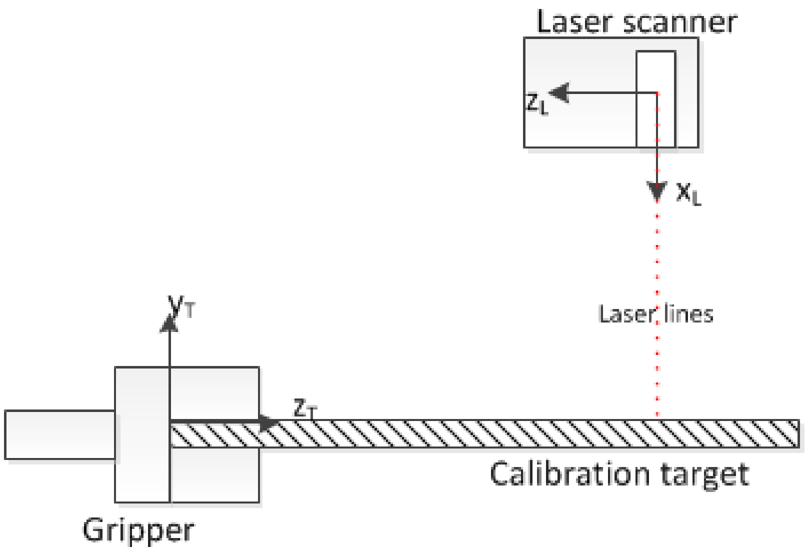

Figure C.2: Frame notation and ideal location. Reprint from [2]

As mentioned previously the calibration target is rigidly attached to the robot tool and the target's frame and the tool's frame is coincident. Finding the orientation and position of the calibration target in the laser scanner's coordinate frame thus gives us the transformation between the sensor and the manipulator.

The surface plane of the target is flat and coincident with the plane spanned by the X_T and Z_T axis. The angle a between the straight and the skewed finger is a design parameter and thus known a priori, as is the distance from the target's frame's origo to the top of the triangle between the two fingers, point A. This distance is arbitrarily named e. The width of the straight finger is also a known design parameter and named w.

C.2.2 Finding the transform

In calculating the pose of the target relative to the scanner, we exploit the fact that the target's surface is parallel to the plane spanned by the X_T and Z_T axis, and movement in these axial directions thus won't affect the distance measured by each of the scanner's laser rays as long as the ray hit the target. Without loss of generality, we assume using the center laser ray for finding the edge.

We can thus easily determine the dimensions of the green triangle on figure C.3 by first positioning the target via the robot, so the center laser line goes between the two fingers of the target. For initial calibration the robot is moved into position by a human operator, while the robot can do this autonomously when re-calibrating. The plane spanned by the target should be placed as close to perpendicular to the laser ray as possible.

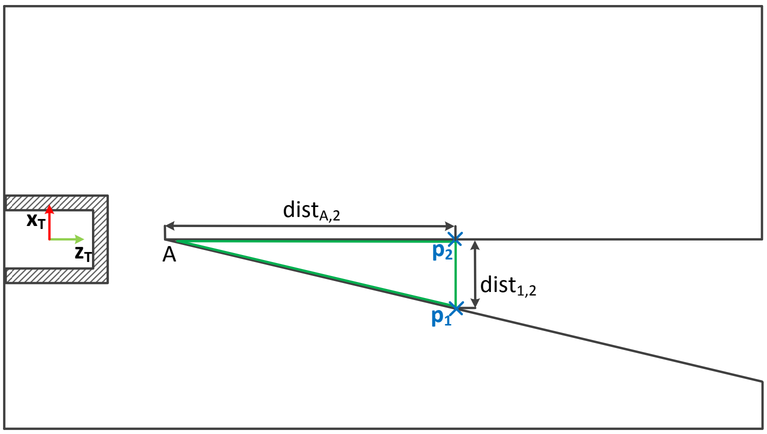

Figure C.3: Finding p_1 and p_2

The target is then moved in the positive X_T axis direction, until the laser detect the inner edge of the target's skew finger. The point in the robot's coordinate frame in which the target intersects with the laser line is denoted

p_1. The target is then moved in the negative X_T axis direction until the inner edge of the target's straight finger is detected by the same laser ray. This point is denoted p_2. We call the distance moved $dist_{1,2}$.

The angle A of the green triangle on Figure C.3 is chosen when designing the target, and the length of the opposite side of the right-angled is the distance between the points p_1 and p_2, which is equal to the distance $dist_{1,2}$. The distance between the triangle's top point A and p_2 is then

$$dist_{A,2} = \frac{\sin\left(\frac{\pi}{2} - A\right) * dist_{1,2}}{\sin(A)} \tag{C.1}$$

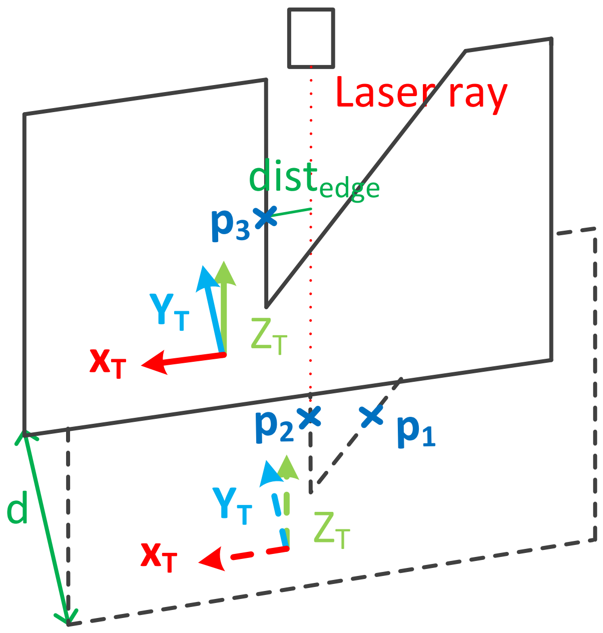

Figure C.4: Pod rotated around Y_T and moved distance d along the Y_T axis

Next the robot moves the target a known distance d in the Y_T axis direction as shown on Figure C.4. If the target is rotated around the Z_T axis, relative to the scanner, the center laser beam will no longer be on the edge of the target after the translation. By moving the target $dist_{edge}$ along the X_T axis until the beam hits the edge once again, the rotation of the target around the Z_T axis in the laser frame Rot_{Z_L} can be found. This point is named p_3 as shown on figure C.4.

$$Rot_{Z_L} = \arcsin\left(\frac{dist_{edge}}{\sqrt{d^2 + {dist_{edge}}^2}}\right) \tag{C.2}$$

The robot then once again moves in the positive X_T axis direction until the laser scanner detect the edge of the target's skew finger, that point is named p_4. The moved distance is named $dist_{3,4}$. As before we can determine the distance from the top point A to p_3

$$dist_{A,3} = \frac{\sin\left(\frac{\pi}{2} - A\right) * dist_{3,4}}{\sin(A)} \tag{C.3}$$

and from the difference between $dist_{A,3}$ and $dist_{A,2}$ determine the rotation of the target around the X_T axis in the laser frame Rot_{Y_L}

$$Rot_{Y_L} = \arcsin\left(\frac{dist_{A,3} - dist_{A,2}}{\sqrt{d^2 + (dist_{A,3} - dist_{A,2})^2}}\right) \tag{C.4}$$

The robot can now correct for the angular differences around the X_T and Z_T axis, so the center laser beam is perpendicular to the plane spanned by the target and thus Y_T and X_L is parallel.

The inner edge of the straight finger is again aligned with the laser beam, and then the target is moved along the Y_T axis until another laser beam hit the outside edge of the target. As the angular distance $dist_{ang}$ between each beam of the scanner and thus also the two beams is known, the measured width w_{meas} of the target can easily be calculated based on the measured distance to the target. Now the rotation of the target around Y_T in the laser frame Rot_{X_L} can be found by comparing the measured width to the known w.

$$Rot_{X_L} = \frac{\pi}{2} - \arcsin\left(\frac{w}{w_{meas}}\right) \tag{C.5}$$

With all the relative rotations known, it is trivial to align all the axes of the target with the axes of the laser scanner and then move the target to a known place in the scanner's field of view, i.e. with the center laser beam intersecting a corner of the target. This directly gives the translational transformation between the scanner and the target, and with the known transformation between both the robot and the target as well as between the target and the laser scanner, it is a simple matter of multiplying the two transformation matrices to get the transformation matrix between robot and laser scanner.

C.3 Result

To verify that the proposed method works, we have tested it with a Motoman MH5L and a Hokuyo URG-04LX laser range scanner, which showed that the method works reliably for determining the transformation between the robot and the laser scanner. The setup can be seen in the left picture on Figure C.5. The calibration target size has been scaled to make it easier to see in the picture.

C.3.1 Accuracy

As part of the reason for developing the method is the missing method for finding the ground truth, it is impossible to determine the actual achieved accuracy. Instead, which is also much more relevant, we evaluate the worst-case accuracy that can be achieved with this setup.

By proper calibration and modeling of the laser scanner, studies like [37] suggest distance measurement errors of $\pm 3\,\mathrm{mm}$ or less. But except for finding the distance from the target to the scanner, all other measurements depends on how accurate an edge can be detected.

To evaluate this, we have moved the target underneath the scanner in $1\,\mu\mathrm{m}$ steps and logged 500 measurements at each position, from which we calculated the mean. A plot of this data can be seen on Figure C.5. Here it can be seen that the falloff of the laser, from when all of the ray is on the target to none is on the target, is 12 μm. If we position the edge in the middle of this falloff, i.e. using a threshold of 90 cm on Figure C.5r, the maximum error is 6 μm.

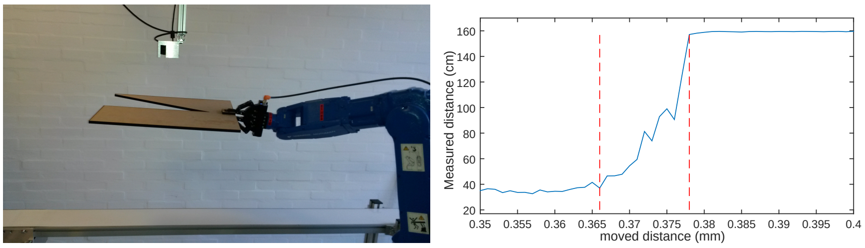

Figure C.5: Left: Test setup with up-scaled calibration target. Right: Distance measured by laser scanner while moving target underneath in 1 μm steps. The red lines denotes a 12 μm interval

As we look for two opposing edges when finding the width of the straight finger w_{meas} for calculating Rot_{X_L} and the distance between the two fingers $dist_{1,2}$ and $dist_{3,4}$ used for calculating Rot_{Y_L}, the error might be up to 12 μm. This interval is marked with red lines on Figure C.5r. On the other hand, it is the same edge we use for determining the last rotation Rot_{Z_L}, thus the error here will theoretically be zero as we just need to find the same point. But as the graph on Figure C.5r is not completely linear, we assume the error of determining $dist_{edge}$ to be $\pm 3\,\mu$m.

For comparison we use the same size calibration target as in [2] with $w = 0.05$ m and $A = 40°$. Note that these values was chosen to give the best possible results in [2]. The maximum error of the rotation Rot_{X_L} is

$$\begin{aligned} err_{Rot_X} &= 90° - \arcsin\left(\frac{w}{w_{meas} + w_{meas_{maxerr}}}\right) \\ &= 90° - \arcsin\left(\frac{0.05}{0.05 \pm 0.000012}\right) = \pm 1.26° \end{aligned} \tag{C.6}$$

If the target is rotated around the X_L axis, the measured width will be higher, meaning the error's ratio will be lower which leads to an even lower error. Increasing the width of the target will also lower the error. Below the effects of these is shown both separately and combined, as $w_{meas} = 0.070711$ for $Rot_{X_L} = \pm 45°$ and $w = 0.05\,\text{m}$

$$\begin{aligned} err_{Rot_{X_{45°}}} &= 45° - \arcsin\left(\frac{0.05}{0.07071 \pm 0.000012}\right) = \pm 0.0097° \\ err_{Rot_{X_{w=0.1}}} &= 90° - \arcsin\left(\frac{0.1}{0.1 \pm 0.000012}\right) = \pm 0.89° \\ err_{Rot_{X_{45°,w=0.1}}} &= 45° - \arcsin\left(\frac{0.1}{0.141421 \pm 0.000012}\right) = \pm 0.0049° \end{aligned} \tag{C.7}$$

We therefore argue that the $err_{Rot_{X_{max}}}$ can easily be lowered to 0.0097° by rotating the target 45° before determining Rot_{X_L}. By changing the size of the target, it can even be lowered to 0.0049° or better.

In calculating the worst case error around Rot_{Y_L}, distances of $dist_{3,4} = 16.8\,\text{mm}$ and $d = 0.1\,\text{m}$ are used in [2]. If $Rot_{Y_L} = 10°$, which it is argued in [2] is the margin of error a human operator can expect to stay within using eyesight, then $dist_{1,2} = 2.02\,\text{mm}$. Based on equations C.1, C.3, and C.4, the error of the calculated rotation, if using $dist_{3,4} = 16.8 \pm 0.012\text{mm}$ and $dist_{1,2} = 2.02 \pm 0.012\text{mm}$ is

$$err_{Rot_{Y_L}} = 10° - \arcsin\left(\frac{dist_{A,3} - dist_{A,2}}{\sqrt{d^2 + (dist_{A,3} - dist_{A,2})^2}}\right) = \pm 0.0159° \tag{C.8}$$

In the interval $[-10°; 10°]$ with steps of 0.0001° the error is found to be between $err_{Rot_{Y_{max}}} = \pm 0.0164°$.

Similar, we can use equation C.2 and the $\pm 3\,\mu\text{m}$ error of $dist_{edge}$ to calculate the worst case error if the true angle $Rot_{Z_L} = 10°$

$$err_{Rot_{Z_L}} = 10° - \arcsin\left(\frac{dist_{edge}}{\sqrt{d^2 + dist_{edge}^2}}\right) = \pm -0.00167° \tag{C.9}$$

and going through the same interval shows that all errors are within $err_{Rot_{Z_{max}}} = \pm 0.00172°$.

With regards to determine the translational error, the worst case error in the Y_L and Z_L is already determined to be $\pm 6\mu$m. According to [37] and most published LIDAR models like that in [34], the error of the laser scanner can be assumed to be Gaussian. Thus, if we sample over K samples, the measured distance error in the X_L direction goes towards 0 as $K \to \infty$ if we beforehand calibrate the scanner to the target to compensate for any offset.

C.4 Discussion

In this paper, we have taken the *two scan* method from [2] and showed how it can be improved by using more scans, but only utilizing a single or two laser beams from the laser scanner. No other paper have attempted to give a metric for the performance of the presented method. In the comparison in Table C.1 it can be seen that while the method from [2] have worst case errors of several degrees, the method presented in this paper have almost no error in the rotations. The translational error is also improved by a factor 166 or better by relying on detecting edges of the calibration target.

Finally it should be noted that these results are dependent on how accurate the calibration target can be manufactured, or rather measured after manufacturing. Especially the mounting on the robot and the transformation between robot and the point A are crucial, but also the angle A, as deviations in this can lead to higher errors in rotation around Y_L and the translational distance Z_L. A robot with purely rotational joints can also introduce small errors when trying to perform straight motion in Cartesian space, especially when the robot operates close to singular positions. The robot used in this paper only has rotational joints though, so the error is marginal. So a robot would obviously be able to perform this calibration better.

It should be noted that all the above observations also holds true for the

Table C.1: Comparison of performance between the two scan method from [2] and the single beam method from this paper

	Two scan	Single beam
X_L (mm)	± 1	0 for $K \to \infty$
Y_L (mm)	± 1	± 0.006
Z_L (mm)	± 1	± 0.006
Rot_{X_L} (°)	± 7.07	± 0.0097
Rot_{Y_L} (°)	$\begin{cases} +2.64 \\ -0.53 \end{cases}$	± 0.0164
Rot_{Z_L} (°)	± 6.8	0.00172

method in [2].

C.5 Conclusion

In this paper we present an effective calibration method that can be utilized directly in the industry. Having an accurate calibration method that can be used autonomously makes it a lot easier to include re-calibration and verification in a daily routine, which can help ensure product quality of manufacturers. This is especially valuable in the food processing industry, where high pressure water is often used for cleaning and can thus easily change the orientation of sensors slightly.

We use a previously presented method for calibration between a laser scanner and a robot manipulator and demonstrates how it can be improved by using only one or two laser beams to handle errors related to quantization of the angles of the laser scanner's beams. It is shown that a LIDAR can detect edges significantly more accurately with much less noise than measure distances, and this is exploited to significantly increase the accuracy of the method with a factor 166 for the translational part and between a factor 32 and almost 4.000 for the rotational calibration.

In determining the accuracy we also show how the worst case error can be determined. This can be useful for quality assurance as it makes it possible to document the repeatability and margin of error of a product produced by sensor controlled robots.

Paper D

Measuring and Modelling Delays in Robot Manipulators for Temporally Precise Control using Machine Learning [1]

Thomas Timm Andersen[1], Heni Ben Amor[2], Nils Axel Andersen[1] and Ole Ravn[1]

[1]Department of Electrical Engineering, Automation and Control Group, Technical University of Denmark, Richard Petersens Plads 326, 2800, Kgs. Lyngby, Denmark

[2]Institute for Robotics and Intelligent Machines, College of Computing, Georgia Institute of Technology, 801 Atlantic Dr NW, Atlanta, GA 30332, USA.

Abstract:
Latencies and delays play an important role in temporally precise robot control. During dynamic tasks in particular, a robot has to account for inherent delays to reach manipulated objects in time. The different types of occurring delays are typically convoluted and thereby hard to measure and separate.

In this paper, we present a data-driven methodology for separating and modelling inherent delays during robot control. We show how both actuation

[1]Reproduced from T. T. Andersen, H. B. Amor, N. A. Andersen, and O. Ravn. “Measuring and Modelling Delays in Robot Manipulators for Temporally Precise Control using Machine Learning”. In: *14th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2015

and response delays can be modelled using modern machine learning methods. The resulting models can be used to predict the delays as well as the uncertainty of the prediction.

Experiments on two widely used robot platforms show significant actuation and response delays in standard control loops. Predictive models can, therefore, be used to reason about expected delays and improve temporal accuracy during control. The approach can easily be used on different robot platforms.

Keywords: Robot control, Automation, Machine learning algorithms

D.1 Introduction

For robots to engage in complex physical interactions with their environment, efficient and precise action generation and execution methods are needed. Manipulation of small objects such as screws and bolts, for example, requires spatially precise movements. However, in dynamically changing environments, spatial precision alone is often insufficient to achieve the goals of the task. In order to intercept a rolling ball on the table, for instance, a robot has to perform *temporally precise* control—the right action command has to be executed at the right time. Yet, by their very nature, actuation commands are never instantaneously executed.

Delays and latencies, therefore, play an important role in temporally precise control and can occur at different locations in the robot control loop. *Actuation delay* is the delay type that most roboticists are aware of. When an action command is sent to the robot's controller, it takes a short while to process the command and calculate the required joint motor input. Imagine a welding robot with an uncompensated actuation delay of 50 ms, working an object on a fairly slow-moving conveyor belt with a speed of 0.5 m/s. The incurred delay would result in a tracking error of 2.5 cm, which could easily destroy a product, or at the very least result in a suboptimal result.

A different type of delay is the *response delay* which measures the amount of time until a real-world event is sensed, processed and updated in memory. Response delay is usually assumed zero, as one would naturally assume that this is sampled and transmitted instantaneously whenever a motion occurs. However, since there is a sampling clock and since the controller also needs some time to pack the data for transmission, the response delay can be a non-negligible amount of time. An important implication of the response delay is the discrepancy between the robot's belief of its own state and the true value of that state. When data is received from the robot, indicating that the robot is at a certain position moving with some velocity, the data is in reality describing a state in the past.

In order to effectively act in dynamic environments and reason about

Figure D.1: Temporally precise control of an industrial robot is realized by modelling the inherent delay in the system. The picture depicts a fast robot movement during data acquisition. Recorded data is processed using machine learning algorithms to generate predictive models for system and response delay.

timing, a robot has to be aware of both the actuation delay as well as the response delay. Sadly, such information is not readily accessible form the robotics company, and no method is currently available for identifying it. This has lead many researches to develop their own controllers, but this is rarely an opportunity for industrial users.

Safety during operation is the most crucial issue for robot controllers, but each robotic company may has different strategies which affect the architecture of the robot controller. It is therefore necessary to consider the controller as a black box from which we must learn the controller-dependent delay characteristics. Direct measurement of these delays is typically difficult, since the different delay types are convoluted and hard to separate. An important challenge is therefore the question of how to separate these two delays as, depending on the executed task, a robot has to compensate for a different type of delay.

In this paper, we present a methodology for measuring and modelling the inherent delays during robot control. We introduce an experimental setup which allows us to collect evidence for both the actuation delay, as well as the response delay. The collected data is then used to learn controller-dependent predictive models of each type of delay. The learned predictive models can be used by a robot to reason about timing and perform temporally precise control.

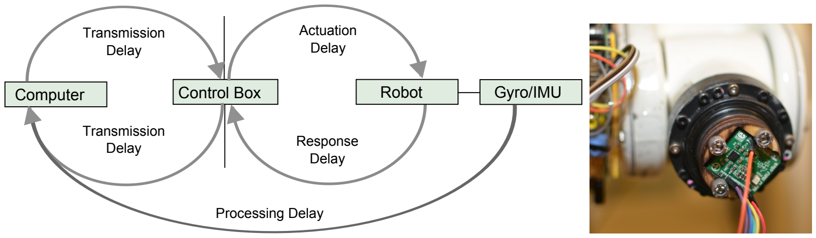

Figure D.2: Left: Delays during the control of a robot manipulator. Transmission delay affects information flow between main control computer and the robot control box. Actuation delay and response delay are introduced in the communication between the control box and the physical robot. Right: For delay modelling an external sensor is mounted, e.g. a gyroscope, to measure discrepancies between command times and execution times.

The contributions of this publication are three-fold. First, we provide a generic method for measuring the actuation and response delay of a robot manipulator. Due to its data-driven nature, the method can be used on a variety of actuators. Second, we show how existing machine learning methods can be used to model and predict the inherent delay. Finally, we show modelling results for two widely used robot platforms, namely the Kuka KR 5 Sixx and the Universal Robots' UR10 robot. The acquired data is made publicly available to the robotics community[38].

D.2 Related work

Modelling time delays is a vital research topic in computer network engineering. In order to ensure fast communication over large computer networks, various models have been put forward to model the mean delay experienced by a message while it is moving through a network [39]. These analytic models typically require the introduction of assumptions, e.g. Kleinrock's independence assumption [40], to make them tractable. Yet, since the network communication is based on a limited number of communication protocols, it is reasonable to use and constantly refine such analytic approaches. Another domain in which latencies and delays play a vital role is Virtual reality (VR). As noted in [41], latencies lead to a sensory mismatch between ocular and vestibular information, can reduce the subjective sense of presence, and most importantly, can change the pattern of behavior such that users make more errors during speeded reaching, grasping, or object tracking. In VR applications, measuring and modelling delays can be very challenging, since the delay can heavily vary based on the involved software components, e.g., rendering engine, as well as highly heterogeneous hardware components, e.g., data gloves, wands, tracking devices etc. In [41] a methodology for estimating delays is presented, which focuses on VR application domains.

In robotics, the delay inherent to control loops can have a detrimental impact on system performance. This is particularly true for sensor-based control used in autonomous robots. Visual servoing of a robot, for example, can be sensitive to the delays introduced through image acquisition and processing [42]. Similarly, delays in proprioception can produce instabilities during dynamic motion generation. In [43], a dynamically smooth controller has been proposed that can deal with delay in proprioceptive readings. However, the approach assumes constant and known time-delay. A major milestone in robot control with time-delay was the ROTEX experiment [44]. Here, extended Kalman filters and graphical representation were used to estimate the state of objects in space, thereby enabling sensor-based long-range teleoperation. How to effectively deal with such communication delays has been a central research question in robotic tele-operation. Delays in robot control loops are not limited to sensor measurements only. A prominent approach for dealing with actuation delays is the Smith Predictor [45]. The Smith Predictor assumes a model of the plant, e.g. robot system, and can become unstable in the presence of model inaccuracies. A different approach has been proposed in [46]. A neural network was first trained to predict the state of mobile robots based on positions, orientations, and the previously issued action commands. The decision making process was, then, based on predicted states instead of perceived states, e.g. sensor readings. The approach presented in our paper follows a similar line of thought. However, instead of predicting specific states of the robot, we are interested in predicting the delay occurring at different parts of the control loop.

D.3 Methodology

In this section, we describe a data-driven methodology for modelling delays in robotic manipulators. We show how to acquire evidence for different types of delays and how this information can be used in conjunction with machine learning methods to produce predictive models for control.

D.3.1 Measuring the delay

The purpose of the presented method is to establish the actuation and response delay that a high-level control program can expect when issuing commands to a robotic controller. To measure these delays, we need to synchronize the issuing of commands with the control loop of the robot controller. To this end, we use the published current state of the robot, which most controllers send out in each control cycle.

The overall system setup which will be used in the remainder of the paper is depicted in Figure D.2 (left). A high-level control program is running on a computer, which sends the commands to the robot control box. The control box, in turn, calculates and issues the low-level commands that drive the robot. The delay between the high-level controller and the control box will be referred to as the *transmission delay*. The transmission delay has already been extensively studied in computer networking [39] and will thus not be treated in this paper. It is particularly crucial in tele-operation scenarios, in which the high-level controller and the robot control box may be separated by thousands of kilometres.

In this paper, however, we focus on the delays incurred between the control box and the robot manipulator. A command that is received by the control box from the high-level program at time $t = 0$ is typically only executed after a delay of ε_1. This is the actuation delay. Similarly, once a command is executed by the robot at time $t = \varepsilon_1$, it takes another delay of ε_2 until the motion is reflected in the controllers memory and transmitted to the high-level program running on the central computer. This is the response delay.

The fundamental idea of our approach is to compare time stamps at the moment a command is issued, the moment the command is executed, and the moment the command gets reflected in the published state of a robot. To this end, it is important to know the ground truth about the true timing of the robot movement. This is realized using an external apparatus in our setup, e.g., a gyroscope or accelerometer, see Figure D.2 (right).

D.3.1.1 Determining ground truth

Since we want to measure the delay of the robot, we need a reliable and accurate method of measuring robot motion. The method needs to measure the current motion without adding a significant delay of its own. This can be achieved by imposing a significantly higher sampling rate than the robot controller.

We use microelectromechanical (MEMS) gyroscopes, or angular rate sensors, for the revolving joints, and MEMS accelerometers for prismatic joints. They offer very high sampling rates of several orders of magnitude higher than many robot controllers publish (e.g. several kHz for affordable sensors), and practically no delay from motion to available measurement. Such sensors cannot readily be used to infer where in the kinematic chain a motion has occurred, hence measurements have to be performed a single joint at a time. Gyroscope measurements often come with significant noise, while accelerometer measurements suffer from drift. However, both of these issues can be compensated for using simple offline filtering in-between measurement and training the model.

D.3.1.2 Acquiring measurements

As mentioned before, our approach is based on comparing time stamps throughout the robot control loop. To this end, we use the published state from the robot as the main sample clock and reference. An experimental trial starts at $t = 0$ upon reception of a first package from the controller. The system time stamp is recorded as soon as data is read, and the byte-encoded package is stored for later parsing to extract the current joint state. Upon reception, a command is sent instructing the robot to start moving a single joint, which we monitor with our angular rate sensor or accelerometer. The commanded movement consists of a rapid acceleration in one direction, followed by a fast deceleration before returning to return to the starting pose. The entire motion trial takes about a second, and all packages received until the robot stands still are stored. Sensor readings from the external sensor are stored by the central computer in order to identify the ground truth time stamp of the moment in which the robot moved.

There are several perturbations that can lead to variations in the incurred delays, in particular physical perturbations. For instance, the force resulting from the gravitational pull on the robot varies with the joint configuration of the robot, just as the direction of motion effects whether the motor needs to work against or along gravity. The different size of motors and gearing in the robot also yields varying results. These perturbations lead to varying static and kinetic friction in the moving parts of a robot. This variation in turn leads to a varying actuation delay.

As the magnitude of the static friction is usually larger than that of the kinetic friction, we assume that the delay is mostly affected by the robot's joint configuration when the motion starts. We assume that the effect by the other joints during a motion after the static friction has been overcome can be neglected. A similar assumption of joint independence if often employed on the joint position controller when using Independent joint control[35].

To acquire a representative data set for modelling delays, we therefore need to map out the delay of each joint for all the different joint combinations, moving in both positive and negative direction. To capture variance in the delay, each combination of joint configuration and direction should be measured several times.

D.3.1.3 Filtering data and computing delays

When extracting the delays, we evaluate the difference between the recorded data. Before doing that, thought, we use a high order low-pass FIR filter (Figure D.3) on the data from the angular rate sensor and correct for any drifting of the accelerometer, based on data recorded while the sensor was held stationary on the robot.

To calculate the delay, we evaluate our two data series generated in each trial; the speed output from the robot controller and the filtered sensor data. The actuation delay is the difference between the moment a command is sent to the robot and the moment a sensor registers the motion, while the response delay is the difference between the moment a sensor registers motion and the moment it is reflected in the robot's current speed data. Both are calculated while taking into account the transmission delay from Figure D.2 (left). Even when filtering out the noise, it can be challenging to establish the exact moment in time when the sensor determines that a motion has started as the measured speed is hardly ever zero. Instead we identify extrema of our recorded data to detect the time difference between the set target speed, the measured speed, and the reported current speed.

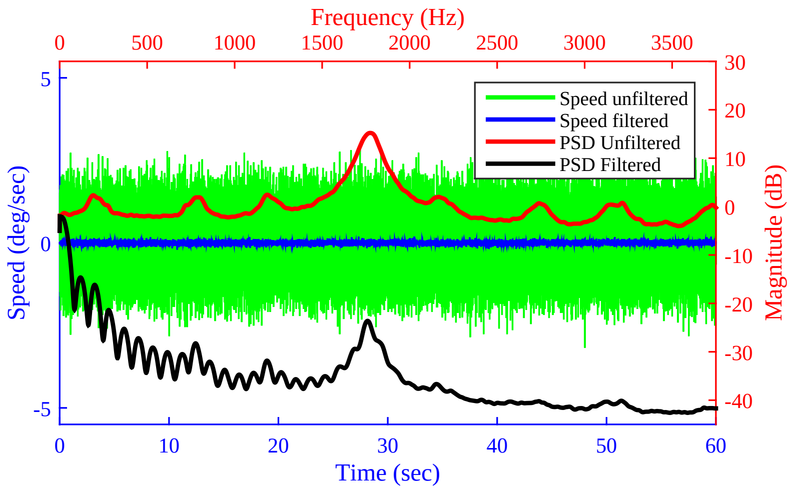

Figure D.3: Gyroscope readings are filtered using a FIR filter. A 60 second datastream (green), recorded without moving the robot, is passed through the filter to remove noise (blue). The frequency component of the data before and after filtering is shown in red and black, calculated using Welch's power spectrum density estimate[47]

D.3.2 Learning Predictive Models of Delay

Next, we want to use the recorded data in order to learn predictive models of robot latencies. Once a predictive model is learned, it can be used by a robot to infer the most likely delay in a given situation. A common approach in robot control is to use a path planner running on the central computer to generate a starting joint configuration and an execution time of the trajectory. To find the actual real time that the robot will use to get to the goal state, we can query the learned predictive models for each moving joint. The individual delay is then added to the execution time of each joint to identify the real execution time.

As input features for the model we use the starting joint configuration of the robot. As mentioned before, forces acting on the robot vary depending on the joint configuration and impact in particular the actuation delay. The output of the model is the expected delay. We learn individual models for the actuation delay and the response delay, since these two delays are unrelated. In line with the assumption of independence between the joints, a separate model is learned for each joint. Introducing the above structured approach, allows for accurate predictions of the delay. To evaluate how a unified model, predicting the delay of all joints, performs, such a model is also trained.

The goal of learning is to generate predictive models that can generalize to new situations and lead to accurate predictions of the expected latencies. To this end, we use three different machine learning methods, namely neural networks (NN) [48], regression trees (RT) [49], and Gaussian processes (GPR) [50]. We use these methods as they can all effectively recover nonlinear relationships between input and output data.

In our specific implementation, we used a feed-forward neural network with 30 neurons in a single hidden layer. Learning was performed using the Levenberg-Marquardt [51] algorithm. In contrast, the regression tree method hierarchically partitions the training data into a set of partitions each of which is modelled through a simple linear model. Both NNs and the RTs produce a single result and do not provide information about the uncertainty in the predicted value. In contrast to that, GPR can learn probabilistic, non-linear mappings between two data sets. Due to the inherent noise and related phenomena, uncertainty handling is a crucial issue when dealing with delays.

By providing the mean and the variance of any prediction, the GPR approach allows us to reason about uncertainty of our prediction. Together, mean and variance form Gaussian probability distribution indicating the expected range of predictions. This information can potentially be exploited to generate upper- and lower-bounds for the expected delays, which is in contrast to both NN and RT.

As both NN, RT, and GPR are well known machine learning methods and we do not add anything to these methods, the theory behind them will not be covered further in this paper.

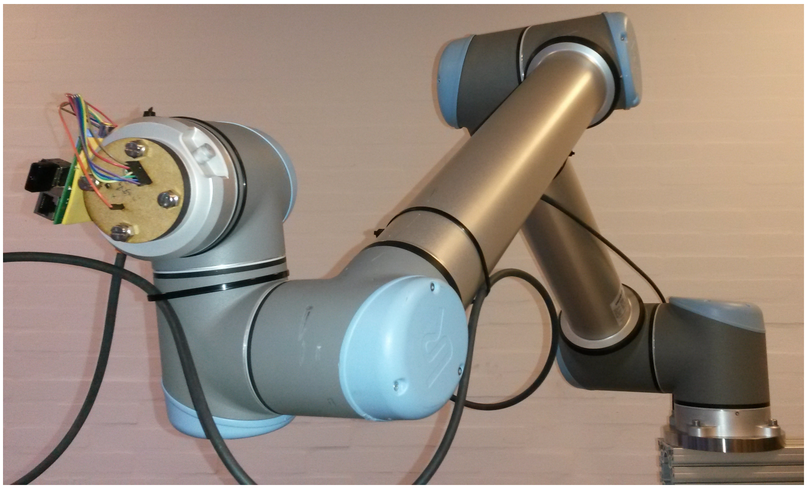

Figure D.4: The Universal Robot UR10 with mounted measuring equipment. The enclosure keeps the sensor at a stable temperature thus avoiding temperature-related drift in measurements.

D.4 Results

D.4.1 Experimental setup

In our experiments, we model the performance of both a Kuka KR 5 sixx (Figure D.1) and a Universal Robot UR10 (Figure D.4). To generate the training data, we mounted a MPU6000 combined angular rate sensor and accelerometer to the end-effector. To avoid temperature-related drifts, we mount the sensor on the robot in an enclosure with low heat conductivity and then let the sensor warm up before measurements. Since these robots only have revolute joints, only the angular rate sensor is used, which outputs data at a rate of 8 kHz.

To collect the data used for training the model, we perform a series of short trials, wherein the robot is commanded to perform a fast acceleration and deceleration motion. For controlling the Kuka robot, the Kuka RSI[15]

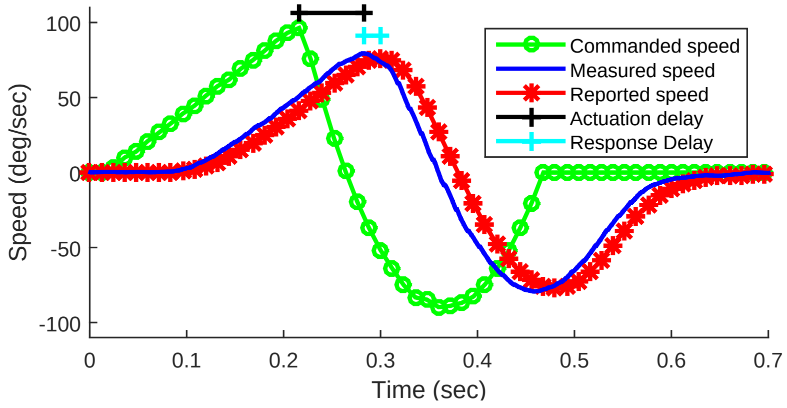

Figure D.5: Typical plot of logged data from a single trial. 33,500 trials were completed on each robot.

protocol is used. It operates with a sample rate of 83.3 Hz (12 ms). As argued in [6], the UR10 is controlled using URScript SpeedJ commands. It operates with a sample rate of 125 Hz (8 ms).

An example trajectory, along with typical outcome of a trial, can be seen on Figure D.5. The plots clearly show a significant time difference in the commanded speed, the reported speed and the measured speed. To capture the variations of the delays, we performed trials on 4 different joints, moving 10 times in both positive and negative direction in 1,920 different joint configurations. A total of 33,500 trials were performed on each robot in order to generate a comprehensive dataset, to be released to the public[38]. For purposes of machine learning, only a subset of the data was later used.

To be able to compare the performance of the two robots, we used the same 1,920 physical joint configuration (i.e. all links vertical) for both robots rather than using the same joint values. This is a necessity since the Denavit-Hartenberg parameters of the robots are not identical and the home position varies, thus positive joint rotation on one robot might lead to negative joint rotation on the other. Sampling only a subspace of the robots' total workspace

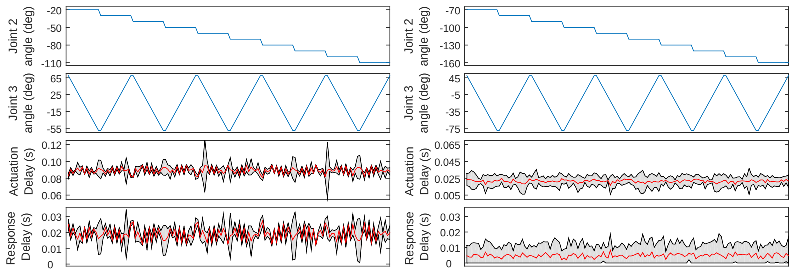

Figure D.6: Actuation and response delay for joint 3 moving in positive direction as a function of varying joint 2 and 3. The red graph is the mean and the gray area is ±2 standard deviations, corresponding to a 95% confidence interval. Note the different y axis interval. Left: Kuka. Right: Universal Robot.

does not introduce bias in the data, but rather limits the model to predict delays within that subspace. By sampling more poses, the model can routinely be extended to cover the entire workspace if needed.

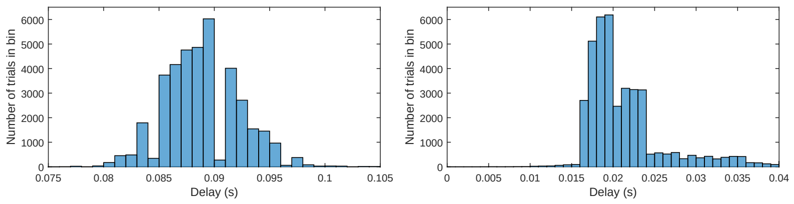

Figure D.7: Combined distribution of the actuation delay of all joints. Note the different x axis interval. Left: Kuka. Right: Universal Robot.

D.4.2 Delay output

As explained in Section D.3.1.3, delays are determined by evaluation of the extrema of the recorded motions. An example of how the delays vary for the two robots can be seen on Figure D.6. The distribution of the actuation delays can be seen on Figure D.7, while a boxplot showing the individual delays per joint is shown on Figure D.8. The same plots for the reaction delays can be seen on Figure D.9 and Figure D.10, respectively.

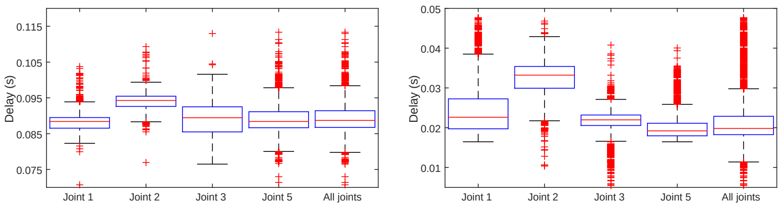

Figure D.8: Boxplot of individual joint's actuation delay. Note the different y axis interval. Left: Kuka. Right: Universal Robot.

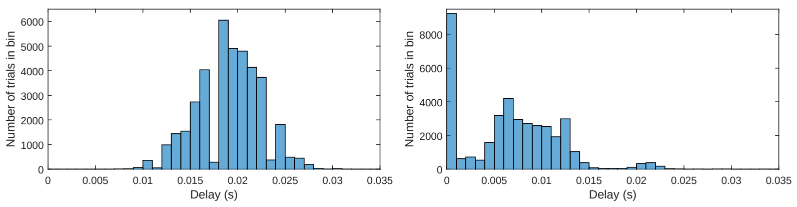

Figure D.9: Combined distribution of the response delay of all joints. Left: Kuka. Right: Universal Robot.

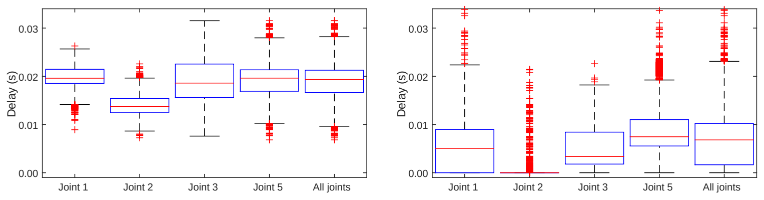

Figure D.10: Boxplot of individual joint's response delay. Left: Kuka. Right: Universal Robot.

D.4.3 Model comparison

The extracted delays were used to train and validate models based on different machine learning algorithms, namely NN, RT, and GPR. For the NN and RT algorithms, we used the standard MatLab implementation, while we used GPstuff[52] for the GPR implementation. The starting joint configuration, the actuated joint, and the rotational direction were used as input. The delays that were measured at each input combination were used for training and testing, using k-fold cross validation with 10 folds to limit overfitting the data and to give an insight on how the model will generalize to an independent dataset. The mean squared error (MSE) from each fold were averaged together and used as a measure of how well the model predicts delays. Models for predicting both the delay of individual joints, as well as a combined model that can predict the delay of all joints were trained. The mean error of each model is derived by taking the square root of the MSE and is shown in Table D.1 and D.2. The tables also shows the resulting mean error if the delay was assumed that of the median of the corresponding boxplots. This gives an indication of the performance of the trained models. Lower values indicate better generalization capabilities, while larger mean error values indicate poor prediction performance.

D.5 Discussion

D.5.1 Evaluating the two robots' delays

As it can be seen on Figure D.7, the actuation delay of the Kuka is significantly higher than on the Universal Robot, even factoring in the higher sample period; the average delay for the Kuka is 7.5 sample periods vs. 2.5 sample periods for the UR. If we relate the figure to the example from the introduction, where a welding robot need to weld an object on a conveyor belt moving at $0.5\,\mathrm{m/s}$, our claim that it is important to compensate for the delay is clearly justified. The Kuka robot would, without compensation, make

Table D.1: Mean error in milliseconds of model fit for actuation delay.

Kuka	Joint 1	Joint 2	Joint 3	Joint 5	Combined	Average
Median	2.27	2.68	4.61	3.51	3.67	3.35
NN	1.79	2.55	4.74	3.37	3.21	3.13
RT	1.99	2.48	3.74	3.76	3.33	3.06
GPR	1.85	2.27	4.30	3.39	3.48	3.06
UR	Joint 1	Joint 2	Joint 3	Joint 5	Combined	Average
Median	6.18	4.64	6.08	2.59	5.33	4.96
NN	4.08	5.32	3.86	2.49	3.12	3.77
RT	4.63	5.41	4.28	2.72	3.42	4.09
GPR	5.01	4.89	3.68	2.47	3.36	3.88

Table D.2: Mean Error in milliseconds of model fit for reaction delay.

Kuka	Joint 1	Joint 2	Joint 3	Joint 5	Combined	Average
Median	2.13	2.37	4.30	3.09	3.33	3.05
NN	1.70	2.32	4.68	2.22	2.36	2.65
RT	1.86	2.21	3.62	2.31	2.37	2.47
GPR	1.63	2.17	4.23	3.11	2.63	2.75
UR	Joint 1	Joint 2	Joint 3	Joint 4	Combined	Average
Median	4.82	2.00	4.08	4.90	5.09	4.18
NN	4.11	2.48	4.24	5.22	4.84	4.01
RT	4.66	2.44	4.47	5.84	5.33	4.35
GPR	3.88	2.44	3.75	5.20	5.05	3.82

a welding seam displaced 4.5 cm ±0.5 cm from the target, while the Universal Robot would miss with 0.75 cm to 1.25 cm.

A deeper look into the actuation delays, which is on Figure D.8, shows that the delays in general only vary with a few ms for each joint. Using our method for measuring the delay and assuming the delay constant at the median of each boxplot would thus decrease the error to within 0.3 cm for a delay within ± 6 ms. If we include the whiskers of the boxplot, corresponding to $\sim \pm 2.7\sigma$ or 99.3% of the data, the worst case error would within 0.85 cm for a delay within ± 17 ms.

Figure D.8 also shows that on both robots, it is joint 2 that has the highest delay. This is the shoulder joint, and the one that lifts the most. This supports our theory that gravity influences the actuation delay. Figure D.10 suggests that the response delay on the other hand is not varying between the joints. This is not surprising, as the response delay, as mentioned previously, is largely incurred by the sampling clock, packing of data and transmission. This most likely happens simultaneously for each joint.

The seemingly correlation between actuation and response delay on Figure D.6 is a consequence of the relatively low temporal resolution of the robot controller data. This is also why it is more dominant on the Kuka robot. As the sum of the actuation and response delay will always be a multiple of the sample period, an actuation delay a few ms below the mean at a specific pose will result in a response delay a few ms above the mean at that pose.

A surprising finding on Figure D.9 is that the response delay for the Kuka robot is more than one sample period, which suggests that sampling and transmission of data takes place in separate sample clock cycles.

D.5.2 Evaluating the models' performance

All of the models are able to predict the delays very accurately to within a mean error of 5 ms and it is thus difficult to say anything conclusive about which model is best. Though all of the models would have a mean error less than 0.35 cm if used for a typical task like welding, which is an improvement of more than a factor 12 for the Kuka robot and almost a factor 3 for the Universal Robot, compared to using the controllers and not assuming any delay. Comparing the learned models with measuring the delay and assuming it to be static shows an improvement between 6 % to 24 %.

The response delay for the Universal Robot shows the least benefit from modelling. This is most likely due to the fact that the spread of the delays are so small. The missing improvement with machine learning is thus a result of the median delay yield a very good guess, and not a result of the models being poor at learning those delays.

It is worth noticing that the mean error in some cases are significantly higher for the Universal Robot models than those of the Kuka robot. This correspond with Figure D.6, where the confidence interval is much broader for the Universal Robot than for the Kuka.

It should also be noted that GPR does not only supply a prediction of the delay, but also outputs a measure of uncertainty, which is not reflected in the tables. For the Universal Robot's large variance, this is certainly an added bonus.

D.6 Conclusion

In this paper we presented a methodology for measuring and separating actuation and response delays in robot control loops. In addition, we introduced a data-driven approach for modelling inherent delays using machine learning algorithms. We showed that the introduced models can be efficiently used to predict occurring delays during temporally precise control.

Real world experiments were used to identify latencies in two widely used robot platforms. The measured delay showed a large potential for improving temporal precision, with more than a factor 12 improvement for one of the robots.

All the employed machine learning algorithms showed similar abilities to further improve the accuracy, with no algorithm showing significantly better accuracy than the others. Still, Gaussian processes seem to be better suited for this task, since they provide a probability distribution over the expected delay. In turn, such a distribution can be used to reason about upper- and lower-bounds in temporal precision.

In our future work we will investigate how inverse models of time delay can be learned. Given a specific time constraint during a control task, an inverse model can be queried for the most appropriate action which will meet the goals of the task while ensuring time constraints.

Bibliography

[1] T. T. Andersen, N. A. Andersen, and O. Ravn. “Exception detection and handling in mission control for mobile robots”. In: *Proceedings of 8th IFAC Symposium on Intelligent Autonomous Vehicles*. Vol. 8. IFAC Proceedings Volumes (IFAC-PapersOnline). Elsevier Science, 2013, pp. 187–192.

[2] T. T. Andersen, N. A. Andersen, and O. Ravn. “Calibration between a laser range scanner and an industrial robot manipulator”. In: *2014 IEEE Symposium on Computational Intelligence in Control and Automation (CICA)*. IEEE. 2014, pp. 1–8.

[3] T. T. Andersen, N. A. Andersen, and O. Ravn. “Optimizing the autonomous self-calibration between a robot and a distance sensor”. In: *Robotics and Computer-Integrated Manufacturing* (2016).

[4] T. T. Andersen, H. B. Amor, N. A. Andersen, and O. Ravn. “Measuring and Modelling Delays in Robot Manipulators for Temporally Precise Control using Machine Learning”. In: *14th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2015.

[5] H. Wu, W. Tizzano, T. T. Andersen, N. A. Andersen, and O. Ravn. “Hand-Eye Calibration and Inverse Kinematics of Robot Arm Using Neural Network”. English. In: *Robot Intelligence Technology and Applications 2*. Ed. by J.-H. Kim, E. T. . Matson, H. Myung, P. Xu, and F. Karray. Vol. 274. Advances in Intelligent Systems and Computing. Springer International Publishing, 2014, pp. 581–591.

[6] O. Ravn, N. A. Andersen, and T. T. Andersen. *UR10 Performance Analysis*. Tech. rep. Technical University of Denmark, Department of Electrical Engineering, 2014.

[7] T. T. Andersen. *Optimizing the Universal Robots ROS driver.* Tech. rep. Technical University of Denmark, Department of Electrical Engineering, 2015.

[8] J. Martin. “Programming real-time computer systems”. In: (1965).

[9] H. Mühe, A. Angerer, A. Hoffmann, and W. Reif. “On reverse-engineering the KUKA Robot Language”. In: *arXiv preprint arXiv:1009.5004* (2010).

[10] A. Elkady and T. Sobh. “Robotics middleware: A comprehensive literature survey and attribute-based bibliography”. In: *Journal of Robotics* 2012 (2012).

[11] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software*. Vol. 3. 3.2. 2009, p. 5.

[12] B. Gerkey, R. T. Vaughan, and A. Howard. “The player/stage project: Tools for multi-robot and distributed sensor systems”. In: *Proceedings of the 11th international conference on advanced robotics*. Vol. 1. 2003, pp. 317–323.

[13] O. Michel. “Webots: Symbiosis between virtual and real mobile robots”. In: *Virtual Worlds*. Springer. 1998, pp. 254–263.

[14] H. Bruyninckx. “Open robot control software: the OROCOS project”. In: *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*. Vol. 3. IEEE. 2001, pp. 2523–2528.

[15] K. R. Group. *KUKA.Ethernet RSI XML 1.1*. KST Ethernet RSI XML 1.1 V1 en. Dec. 2007.

[16] A. B. Beck, N. A. Andersen, J. C. Andersen, and O. Ravn. “Mobotware- a plug-in based framework for mobile robots”. In: *Proceedings IFAC symposium on intelligent autonomous vehicles*. Vol. 7. 2010.

[17] Z.-h. Duan, Z.-x. Cai, and Y. Jin-xia. “Fault diagnosis and fault tolerant control for wheeled mobile robots under unknown environments: A survey”. In: *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE. 2005, pp. 3428–3433.

[18] D. Marco, A. Healey, and R. B. McGhee. *Autonomous underwater vehicles: Hybrid control of mission and motion*. Springer, 1996.

[19] F. Zhou, J. Wang, Y. Li, J Wang, and H. Xiao. “Control of an inspection robot for 110KV power transmission lines based on expert system design methods”. In: *Control Applications, 2005. CCA 2005. Proceedings of 2005 IEEE Conference on*. IEEE. 2005, pp. 1563–1568.

[20] E. J. Friedman-Hill et al. “Jess, the java expert system shell”. In: *Distributed Computing Systems, Sandia National Laboratories, USA* (1997).

[21] R. C. Arkin. “Towards the unification of navigational planning and reactive control”. In: (1989).

[22] D. H. V. Sincák. “Multi–robot control system for pursuit–evasion problem”. In: *Journal of Electrical Engineering* 60.3 (2009), pp. 143–148.

[23] R. Grove. “Internet-based expert systems”. In: *Expert systems* 17.3 (2000), pp. 129–135.

[24] R. Orchard. “Fuzzy reasoning in Jess: the Fuzzy J toolkit and Fuzzy Jess”. In: (2001).

[25] S. Ong, N An, and A. Nee. “Web-based fault diagnostic and learning system”. In: *The International Journal of Advanced Manufacturing Technology* 18.7 (2001), pp. 502–511.

[26] R. N. Jørgensen, M. Nørremark, C. G. Sørensen, and N. A. Andersen. “Utilising scripting language for unmanned and automated guided vehicles operating within row crops”. In: *computers and electronics in agriculture* 62.2 (2008), pp. 190–203.

[27] K. G. Understrup. *Laserbaseret forhindringsundvigelse*. Jan. 2011. URL: `http://aut.elektro.dtu.dk/mobotware/doc/auavoidk.pdf`.

[28] M. E. Antone and Y. Friedman. “Fully Automated Laser Range Calibration.” In: *BMVC*. 2007, pp. 1–10.

[29] M. Hvilshøj, S. Bøgh, O. Madsen, and M. Kristiansen. “Calibration techniques for industrial mobile manipulators: theoretical configurations and best practices”. In: *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*. VDE. 2010, pp. 1–7.

[30] A. M. McIvor. “Calibration of a laser stripe profiler”. In: *3-D Digital Imaging and Modeling, 1999. Proceedings. Second International Conference on*. IEEE. 1999, pp. 92–98.

[31] Q. Zhang and R. Pless. “Extrinsic calibration of a camera and laser range finder (improves camera calibration)”. In: *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*. Vol. 3. IEEE. 2004, pp. 2301–2306.

[32] C. Mei and P. Rives. “Calibration between a central catadioptric camera and a laser range finder for robotic applications”. In: *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. IEEE. 2006, pp. 532–537.

[33] V. Pradeep, K. Konolige, and E. Berger. “Calibrating a multi-arm multi-sensor robot: A bundle adjustment approach”. In: *Experimental Robotics*. Springer. 2014, pp. 211–225.

[34] S. Thrun, W. Burgard, D. Fox, et al. *Probabilistic robotics*. Vol. 1. MIT press Cambridge, 2005.

[35] M. W. Spong, S. Hutchinson, and M. Vidyasagar. *Robot modeling and control*. John Wiley & Sons New York, 2006.

[36] H. A. C. LTD. *Scanning Laser Range Finder URG-04LX Specifications*.

[37] L. Kneip, F. Tâche, G. Caprari, and R. Siegwart. “Characterization of the compact Hokuyo URG-04LX 2D laser range scanner”. In: *Robotics and Automation, 2009. ICRA’09. IEEE International Conference on*. IEEE. 2009, pp. 1447–1454.

[38] http://aut.elektro.dtu.dk/staff/ttan/delay.html.

[39] A. S. Tanenbaum and D. J. Wetherall. *Computer Networks*. 5th. Prentice Hall, 2011.

[40] A. Popescu and D. Constantinescu. “On Kleinrock’s Independence Assumption”. In: *Network Performance Engineering*. Ed. by D. Kouvatsos. Vol. 5233. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 1–13.

[41] M. Di Luca. “New Method to Measure End-to-end Delay of Virtual Reality”. In: *Presence: Teleoper. Virtual Environ.* 19.6 (Dec. 2010), pp. 569–584.

[42] A. Koivo and N. Houshangi. “Real-time vision feedback for servoing robotic manipulator with self-tuning controller”. In: *Systems, Man and Cybernetics, IEEE Transactions on* 21.1 (Jan. 1991), pp. 134–142.

[43] S. Bahrami and M. Namvar. “Motion tracking in robotic manipulators in presence of delay in measurements”. In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. May 2010, pp. 3884–3889.

[44] G. Hirzinger, K. Landzettel, and C. Fagerer. “Telerobotics with large time delays-the ROTEX experience”. In: *Intelligent Robots and Systems ’94. ’Advanced Robotic Systems and the Real World’, IROS ’94. Proceedings of the IEEE/RSJ/GI International Conference on*. Vol. 1. Sept. 1994, 571–578 vol.1.

[45] P. Welch. “A controller to overcome dead time”. In: *ISA Journal* 6.2 (1959), pp. 28–33.

[46] S. Behnke, A. Egorova, A. Gloye, R. Rojas, and M. Simon. “Predicting Away Robot Control Latency”. English. In: *RoboCup 2003: Robot Soccer World Cup VII*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, pp. 712–719.

[47] P. Welch. “A direct digital method of power spectrum estimation”. In: *IBM Journal of Research and Development* 5.2 (1961), pp. 141–156.

[48] C. M. Bishop. *Neural Networks for Pattern Recognition*. New York, NY, USA: Oxford University Press, Inc., 1995.

[49] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984.

[50] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2005.

[51] D. W. Marquardt. "An algorithm for least-squares estimation of nonlinear parameters". In: *SIAM Journal on Applied Mathematics* 11.2 (1963), pp. 431–441.

[52] J. Vanhatalo, J. Riihimäki, J. Hartikainen, P. Jylänki, V. Tolvanen, and A. Vehtari. "GPstuff: Bayesian modeling with Gaussian processes". In: *The Journal of Machine Learning Research* 14.1 (2013), pp. 1175–1179.

Technical University of Denmark
Automation and Control (AUT)
Richard Petersens Plads 326
DK-2800, Kgs. Lyngby
Denmark
Phone: (+45) 45 25 35 76
Email: info@elektro.dtu.dk
www.elektro.dtu.dk