Bringing a Humanoid Robot Closer to Human Versatility: Hard Realtime Software Architecture and Deep Learning Based Tactile Sensing

Berthold Bäuml

Kumulative Dissertation
zur Erlangung des Grades eines
Doktors der Ingenieurwissenschaften – Dr.-Ing. –

Vorgelegt im Fachbereich 3 (Mathematik und Informatik)
Universität Bremen

2. Dezember 2018

Datum des Promotionskolloquiums: 22. Januar 2019

Gutachter
Prof. Dr. Bernd Krieg-Brückner (Universität Bremen)
Prof. Dr. Gerd Hirzinger (TU München, DLR)

Abstract

For centuries, it has been a vision of man to create humanoid robots, i.e., machines that not only resemble the shape of the human body, but have similar capabilities, especially in dextrously manipulating their environment. But only in recent years it has been possible to build actual humanoid robots with many degrees of freedom (DOF) and equipped with torque controlled joints, which are a prerequisite for sensitively acting in the world.

In this thesis, we extend DLR's advanced mobile torque controlled humanoid robot Agile Justin into two important directions to get closer to human versatility. First, we enable Agile Justin, which was originally built as a research platform for dextrous mobile manipulation, to also be able to execute complex dynamic manipulation tasks. We demonstrate this with the challenging task of catching up to two simultaneously thrown balls with its hands. Second, we equip Agile Justin with highly developed and deep learning based tactile sensing capabilities that are critical for dextrous fine manipulation. We demonstrate its tactile capabilities with the delicate task of identifying an objects material simply by gently sweeping with a fingertip over its surface.

Key for the realization of complex dynamic manipulation tasks is a software framework that allows for a component based system architecture to cope with the complexity and parallel and distributed computational demands of deep sensor-perception-planning-action loops – but under tight timing constraints. This thesis presents the communication layer of our aRDx (agile robot development – next generation) software framework that provides hard realtime determinism and optimal transport of data packets with zero-copy for intra- and inter-process and copy-once for distributed communication.

In the implementation of the challenging ball catching application on Agile Justin, we take full advantage of aRDx's performance and advanced features like channel synchronization. Besides developing the challenging visual ball tracking using only onboard sensing while "everything is moving" and the automatic and self-contained calibration procedure to provide the necessary precision, the major contribution is the unified generation of the reaching motion for the arms. The catch point selection, motion planning and the joint interpolation steps are subsumed in one nonlinear constrained optimization problem which is solved in realtime and allows for the realization of different catch behaviors.

For the highly sensitive task of tactile material classification with a flexible pressure-sensitive skin on Agile Justin's fingertip, we present our deep convolutional network architecture TactNet-II. The input is the raw 16000 dimensional complex and noisy spatio-temporal tactile signal generated when sweeping over an object's surface. For comparison, we perform a thorough human performance experiment with 15 subjects which shows that Agile Justin reaches superhuman performance in the high-level material classification task (What material id?), as well as in the low-level material differentiation task (Are two materials the same?). To increase the sample efficiency of TactNet-II, we adapt state of the art deep end-to-end transfer learning to tactile material classification leading to an up to 15 fold reduction in the number of training samples needed.

The presented methods led to six publication awards and award finalists and international media coverage but also worked robustly at many trade fairs and lab demos.

Zusammenfassung

Schon seit Jahrhunderten ist es eine Vision des Menschen, humanoide Roboter zu bauen, d.h. Maschinen, die nicht nur die Form des menschlichen Körpers nachahmen, sondern auch ähnliche Fähigkeiten gerade im der geschickten Manipulation ihrer Umwelt haben. Aber erst in den letzten Jahren konnten tatsächliche humanoide Roboter mit vielen Freiheitsgraden und mit drehmomentgeregelten Gelenken gebaut werden. Drehmomentregelung ist dabei eine Grundvoraussetzung für das feinfühlige Agieren in der Welt.

In dieser Arbeit erweitern wir den fortgeschrittenen, mobilen und drehmomentgeregelten humanoiden Roboter des DLR, Agile Justin, in zwei wesentliche Richtungen. Erstens erreichen wir, dass Agile Justin, der ursprünglich als Forschungsplattform für geschickte mobile Manipulation entwickelt wurde, nun auch komplexe dynamische Manipulationsaufgaben ausführen kann. Dies wird an Hand der schwierigen Aufgabe, zwei gleichzeitig geworfene Bälle mit den Händen zu fangen, gezeigt. Zweitens statten wir Agile Justin mit einem hochentwickelten und auf Deep Learning basierten taktilen Sinn aus, der entscheidend für die geschickte Feinmanipulation ist. Wir demonstrieren seine taktilen Fähigkeiten an Hand einer Aufgabe, die grosse Feinfühligkeit erfordert: das Erkennen des Materials, aus dem ein Objekt besteht, alleine indem man sanft mit dem Finger darüberstreicht.

Grundvoraussetzung für die Umsetzung von komplexen, dynamischen Manipulationsaufgaben ist ein Software Framework, das trotz der engen zeitlichen Randbedingungen eine komponentenbasierte Systemarchitektur ermöglicht. Denn nur eine solche komponentenbasierte Architektur erlaubt es, mit der Komplexität und dem hohen Bedarf an paralleler und verteilter Rechenleistung einer tiefen Sensor-Wahrnehmungs-Planungs-Aktions Schleife umzugehen. Die vorliegende Arbeit stellt die Kommunikationsschicht unseres aRDx[1] Software Frameworks vor. Diese Kommunikationsschicht bietet harte Echtzeit und optimalen Transport von Datenpaketen mit zero-copy Semantik für Intra- und Interprozesskomunikation und copy-once Semantik für verteilte Kommunikation.

Für die Implementierung des Ballfang-Szenarios mit Agile Justin nutzen wir die Performanz und alle fortgeschrittenen Funktionen von aRDx, wie z.B. die Synchronisierung von Kommunikations-Channels. Neben der Entwicklung der anspruchsvollen visuellen Ballverfolgung, die nur Onboard-Sensoren nutzt, obwohl älles in Bewegung ist", und eines Verfahrens zur Autokalibration für den multisensoriellen Oberkörpers, um die notwendige Genauigkeit zu erreichen, ist der wesentliche Beitrag die vereinheitlichte Generierung der Fangbewegung für die Arme. Die sonst üblichen Einzelschritte, Wahl des Abfangpunktes, Bewegungsplanung und Gelenkwinkelinterpolation, werden in zu einem nichtlinearen Optimierungsproblem mit Randbedingungen zusammengefasst und in Echtzeit gelöst. Dieses Verfahren erlaubt es ausserdem unterschiedliches Fangverhalten zu erzeugen.

Für die Umsetzung der feinfühlige Aufgabe, Materialien alleine mit Hilfe des taktilen Signals der drucksensiblen Haut auf der Fingerspitze von Agile Justin zu erkennen, beschreiben wir unsere Deep Convolutional Network Architektur TactNet-II. Als Input für TactNet-II wird das unverarbeitete 16000 dimensionale, komplexe und verrauschte raumzeitliche taktile Signal verwendet, das beim Streichen über eines Objektes entsteht. Für

[1] Agile Robot Development – Next Generation

einen Vergleich der erreichte Erkennungsgenauigkeit mit dem Menschen zu haben, haben wir ein umfangreiche Klassifikationsexperimente mit 15 Testpersonen durchgeführt. Dabei hat sich gezeigt, dass Agile Justin höhere Erkennungsraten als der Mensch erreicht, sowohl bei der eher kognitiven Aufgabe der Materialerkennung als auch bei der eher sensornahen Materialunterscheidung. Um die Effizienz von TactNet-II bzgl. der Anzahl der benötigten Trainingsdaten zu erhöhen, haben wir aktuelle Verfahren für das Deep End-to-End Transferlernen für TactNet-II angepasst. Wir erreichten dadurch eine bis zu 15-fache Reduzierung der benötigten Trainingsdaten.

Die hier vorgestellten Methoden haben zum einen zu sechs Publikationspreisen und Preisnominierungen geführt, konnten aber auch robust bei vielen Messen und Labordemonstrationen gezeigt werden.

Contents

Chapter 1

Introduction

1.1 Why building humanoid robots?

Building humanoid robots has been a fascinating vision for centuries. Humanoid robots are machines which not only resemble the physical shape of the human body but should also have similar capabilities in autonomously acting in their environment, especially in dextrous manipulation.

Besides the fascination, there are also rational reasons to do research in humanoid robots:

- By building machines that try to reach human capabilities, we can get insights about ourselves as humans: how does the human body mechanically and sensorially work and how does human intelligence at all cognitive levels work. We can also learn about how hard the tasks really are the human seems to master so effortlessly.

- A robot with human capabilities would be the ultimate versatile tool to free the human from dangerous, hard or tedious tasks. Imagine humanoid robots to work as Robonauts in the inhospitable outer space, to be co-workers in industrial production or to serve as assistants at home in the household.

- Humanoid robots are by far the most complex robotic systems and building such robots drives the technological developments in diverse areas as mechanics, sensors and actuators, control, software architectures, and up to machine learning and artificial intelligence.

It is only in the last few decades that technology has evolved far enough to actually work on the realization of humanoid robots. One obvious challenge in humanoid robotics is bipedal walking. In 1996 the Honda P2 was presented as the first practical humanoid robot with two arms and legs and a, somehow, human-like walking pattern. In 2006, Honda demonstrated with ASIMO the first humanoid robot which could run (6 km/h). The currently most advanced humanoid robot with regard to bipedal locomotion is Boston Dynamics' amazing Atlas robot [Boston Dynamics, 2018] which not only can run in rough terrain, but jump and even handle a challenging obstacle course using the dynamics of its whole-body.

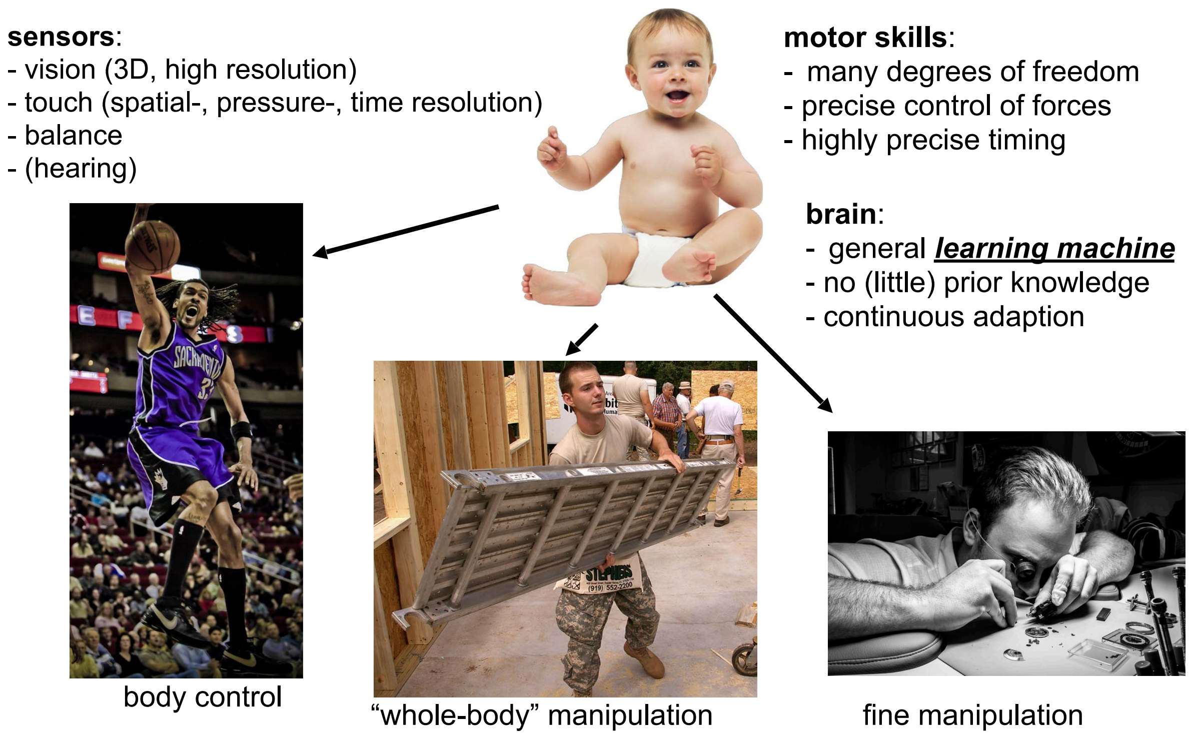

sensors:
- vision (3D, high resolution)
- touch (spatial-, pressure-, time resolution)
- balance
- (hearing)
motor skills:
- many degrees of freedom
- precise control of forces
- highly precise timing
brain:
- general ***learning machine***
- no (little) prior knowledge
- continuous adaption
body control
"whole-body" manipulation
fine manipulation

Figure 1.1: Example tasks for human versatility in dextrous manipulation and the system pre-requisites for their realization. A human learns most of this skills autonomously by interacting with its environment. [1]

1.2 Human Versatility in Dextrous Manipulation

Even more characteristic and challenging than bipedal walking is the human's capability for dextrous manipulation, i.e., humans can interact with and shape their environment using their articulated arms and hands.

Fig. 1.1 depicts three challenging tasks that show off the human versatility

High-performance sports with whole-body control with many degrees of freedom (DOF) and with a timing precision in the millisecond range.

Construction work with detailed control of forces in whole-body manipulation.

Watchmaking which is a delicate fine manipulation task with the fingers.

[1] Image sources: "Baby" from `http://pngimg.com/uploads/baby/baby_PNG17911.png` (CC BY-NC 4.0); "Basketball" from `https://pxhere.com/en/photo/564524` (Creative Commons CC0); "Worker" from `http://archive.defense.gov/photoessays/photoessaySS.aspx?id=1775` ("The appearance of U.S. Department of Defense visual information does not imply or constitute DOD endorsement."); "Watchmaker" from `https://www.flickr.com/photos/richardb23/16880266095` (CC BY-NC-ND 2.0).

To be able to perform such a broad range of challenging tasks, the following prerequisites that go far beyond a classical industrial robot with a position controlled arm and a two finger gripper are required.

Motor skills

- Many degrees of freedom, especially, articulated arms and hands.
- Precise control of the forces when in contact with the environment, e.g., when manipulating objects.
- Precise timing in the actuation over all degrees of freedom.

Sensor skills

- High-resolution 3D vision for modeling the environment.
- Force/torque sensing in all DOF to enable force control.
- High-resolution tactile sensing with millimeter spatial and milliseconds temporal resolution, especially at the finger tips, for dextrous fine manipulation.

Information processing

- High ″computational power″ (the brain) for sensor signal processing, geometrical and semantic environment modeling, whole-body motion planning and up to cognitive reasoning.
- Precise timing in the millisecond range despite distributed sensor, actor and computing resources.
- Learning capabilities as a prerequisite for autonomously and robustly acting in complex and ever changing environments.

1.3 Agile Justin: A Versatile Mobile Humanoid Robot

Over the past 10 years, at DLR we developed and continuously upgraded a family of torque controlled mobile humanoid robots[2]. With the latest key extensions towards complex dynamic tasks and advanced tactile sensing, which we both present in this thesis, our wheeled humanoid robot ″Agile Justin″ fulfills all of the above listed prerequisites for human versatility in dextrous manipulation.

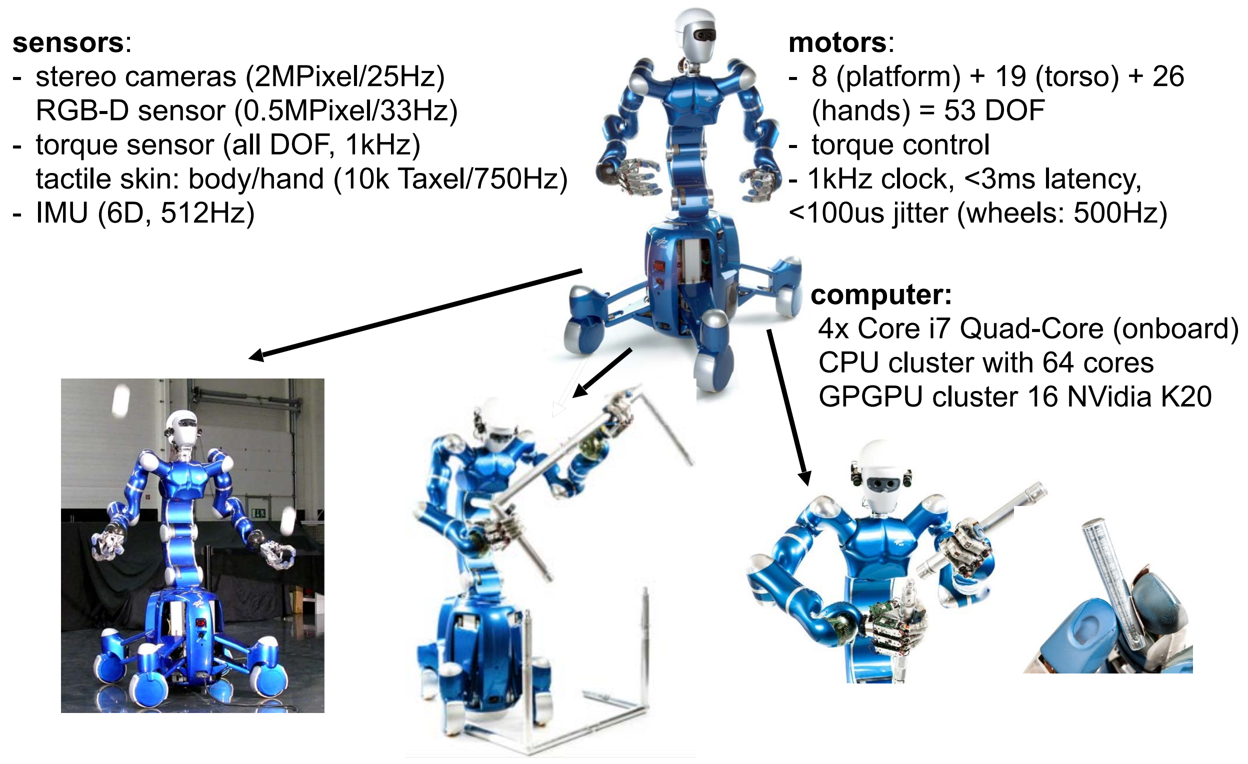

sensors:
- stereo cameras (2MPixel/25Hz)
RGB-D sensor (0.5MPixel/33Hz)
- torque sensor (all DOF, 1kHz)
tactile skin: body/hand (10k Taxel/750Hz)
- IMU (6D, 512Hz)
motors:
- 8 (platform) + 19 (torso) + 26 (hands) = 53 DOF
- torque control
- 1kHz clock, <3ms latency, <100us jitter (wheels: 500Hz)
computer:
4x Core i7 Quad-Core (onboard)
CPU cluster with 64 cores
GPGPU cluster 16 NVidia K20

Figure 1.2: System overview of the advanced mobile humanoid robot Agile Justin including the extension presented in this thesis. Also depicted are example scenarios, roughly resembling those in Fig. 1.1 for the human, showing Agile Justin comes close to human versatility in dextrous manipulation.

1.3.1 System Overview

Fig. 1.2 gives an overview of Agile Justin's current motor and sensor skills and its information processing capabilities. We describe these in more detail below.

Mechatronics: Agile Justin's upper body [Ott et al., 2006] is based on two DLR lightweight arms (2x 7 DOF, 10 kg payload each) [Hirzinger et al., 2002], two DLR Hand-II [Butterfass et al., 2001] (4x3+1 DOF, 20 N force at fingertip), a torso with 3 DOF, and a multi-sensor head with a 2 DOF neck. All DOF are actuated and except for the neck joints and the hands' palm reconfiguration motor, all DOF are torque controlled in a common con-

[2]The family of torque controlled humanoid robots started with *Justin* [Ott et al., 2006], a stationary humanoid upper body. Later, a mobile platform was added which led to the name *Rollin' Justin* [Borst et al., 2009]. *Agile Justin* was initially built as a clone of Rollin' Justin, only with faster joints and a more performant mobile platform (hence the name "Agile"). It is only later with the extensions we describe in this thesis that Agile Justin comes close to human versatility. Agile Justin's capabilities exceed those of all other members of the Justin family. Therefore, when we report about tasks performed with any of the Justin's, we simply say that Agile Justin performed them – because he could do so as well.

trol loop with 1 kHz rate, a communication latency of <3 ms and jitter of $< 100\,\mu$s. The omnidirectional mobile platform [BORST et al., 2009] has 8 DOF which (since the latest upgrade) are also torque controlled with a sample rate of 500 Hz. This allows the system to perform well coordinated motions from the wheels to the fingertips.

3D Perception: The multi-sensor head is equipped with a pair of 2 megapixel stereo cameras as well as an IMU (inertial measurement unit) and, for advanced 3D perception, an RGB-D sensors (Microsoft Kinect). Based on the Kinect data, a GPU-based mapping algorithm generates in realtime dense 3D models of the whole workspace with a resolution of 2mm. The high-quality models are amongst others used as the basis for object recognition and pose estimation.

Whole body motion planning: For fast motion planning in the self-acquired 3D environment models and for all 22 DOF of Justin's torso and mobile platform, an optimization-based planning (OMP) method is used. Besides being fast, especially in replanning of trajectories, optimization-based planning allows to naturally incorporate dynamic constraints and objectives.

Tactile sensing: The articulated hands are equipped with a sensitive tactile skin with a high spatial and temporal resolution (750 Hz and 2 mm at the finger tips). The complex and noisy spatio-temporal signal is processed with advanced deep learning methods, so that Agile Justin can, for example, identify the material an object is made of by simply sweeping over its surface or precisely sense and control the slippage of grasped objects. In addition, the whole mobile platform is covered by a highly sensitive elastic tactile skin [KREYSSIG, 2016] which, for example, allows the robot to move safely in confined environments.

Auto-calibration: The multi-sensorial upper body is calibrated completely automatically and without any external tool, including the intrinsic and extrinsic parameters for the stereo cameras, RGB-D camera, IMU, joint elasticities, and offsets. In addition, fast calibration procedures for the sensors of the mobile base and the mounted tactile skin are provided. Fast and automatic calibration is essential, because for advanced perception methods the precise spatial and temporal relations of the sensors have to be known. However, due to the lightweight structure of the robot as well as inevitable maintenance work, the sensor relations have to be recalibrated all the time.

Computational resources: Agile Justin is equipped with four Core i7 Quad-Core boards in the mobile platform and wirelessly coupled external resources including a GPGPU cluster with 16 Nvidia K20 GPUs for realtime 3D modeling and deep learning and a Xeon CPU cluster with 64 processor cores for parallel optimization-based motion planning. As a research platform, easy scalability of compute resources is important and therefore only components which require high rates and low jitter, e.g., the advanced whole-body control algorithms, or high communication bandwidth, e.g., image processing for the stereo cameras, run onboard, everything else runs on the remote servers.

Software framework: The software architecture of Agile Justin is based on our robotic framework aRDx (Agile Robot Development – Next Generation) we developed for research in mobile manipulation and robot learning on complex and performant robotic systems. The low-level communication layer of aRDx is highly performant and hard realtime capable. It allows for detailed control of the quality of service and optimally transports data packets for intra-process, inter-process (zero-copy) as well as networked (copy-once) communication. This allows Agile Justin's fast and deep sensor-perception-planning-action loop to span multiple computers, even including the GPGPU server cloud in a remote building, with a timing precision in the millisecond range.

1.3.2 Experimental Scenarios

Agile Justin has proven to successfully perform in challenging experimental manipulation scenarios (see Fig. 1.2) which roughly resemble those shown in Fig.1.1 for the human. Here we describe the scenarios in some detail.

Playing ball: In this demanding benchmark scenario [7], up to two balls are thrown towards the robot and it has to catch them with its hands using only onboard sensing. This demands for fast 3D perception, dynamical whole-body motion planning and precise (spatial and temporal) execution of the motion over all DOF (mobile platform, torso, arms and fingers). Agile Justin can not only catch a ball but even throw it back again using a coordinated motion of all its DOF.

Building a scaffold structure: This scenario demands for dextrous as well as whole-body manipulation with detailed control of the forces exerted on the objects. In addition, fast 3D modeling and interpretation of the geometrically complex environment in combination with fast motion planning is required. Our longterm vision is to enable the robot not only to autonomously execute the construction task, but to acquire the necessary skills through autonomous learning.

Fine manipulation and tactile material classification: The tactile skin on the finger tips with its high sensitivity and high spatio-temporal resolution not only allows for fine manipulation, e.g., by sensing the orientation of a small grasped object. By processing the spatio-temporal signal of the skin it is possible to discriminate objects by their material – a skill which is, e.g., important when the objects would be indistinguishable from their 3D shape alone. To do so, the robot compares the data obtained by gently sweeping its fingers over the object with previously learned classes using advanced deep learning methods.

In summary, this scenarios show that, due to the extensions presented in this thesis, Agile Justin already comes close to human dexterity and versatility in mobile manipulation with regard to sensor and motor skills as well as fundamental perception and planning. It is therefore an almost ideal platform for research in intelligent autonomous mobile manipulation where, for the first time, progress in cognitive capabilities is no longer hindered by the underlying robotic system. Or in other words: there is no reason that Agile Justin could not perform tasks similar to a human – it is now all about making the robot more intelligent.

1.4 Related Work

As stated in [Ackerman, 2014], Agile Justin is "arguably one of the most, if not the most, capable dual-armed mobile humanoid robots in existence". In what follows, we give an overview of the state of the art of other advanced mobile humanoid robots and their capabilities in comparison to Agile Justin.

Dynamic Capabilities

There are only few mobile humanoid robots that are dynamically more capable than Agile Justin. Outstanding is Boston Dynamics' legged humanoid robot Atlas [Boston Dynamics, 2018] which can even run and jump in a challenging obstacle course. But Atlas has only 6 DOF arms which are not precisely controllable due to the hydraulic actuation. Although it can be equipped with the articulated Sandia Hands [Labs, 2012], the robot only showed off simple manipulation tasks like moving boxes or turning valves.

Also the family of humanoid robots from Sarcos, like the CB [Cheng et al., 2006], is based on hydraulic actuation which allows for the realization of dynamic tasks like catching a ball or juggling [Riley and Atkeson, 2002], but lacks from precise controllability which hinders the execution of dextrous manipulation tasks.

Agile Justin is up to now the only mobile humanoid robot that can perform such dynamically challenging tasks like ball catching and at the same time has the precision and sensitivity to execute dextrous manipulation tasks.

Torque Control

Justin was the first humanoid robot with torque control in all joints [Borst et al., 2007]. As the precise control of the forces the robot exerts on its environment is essential for dextrous manipulation, many recent advanced mobile humanoids are now torque controlled as well. E.g., KIT's family of humanoids including the legged Armar-4 [Asfour et al., 2013] and the recently presented wheeled Armar-6 [Asfour et al., 2018], DLR's legged TORO [Englsberger et al., 2014], or the commercially available TALOS [PAL Robotics, 2018] from PAL robotics . But not all humanoid robots have dedicated torque sensors in each joint, which limits the control accuracy like in the humanoid open-source platform iCub [Metta et al., 2008] or the wheeled humanoids TWENDY-ONE [Iwata and Sugano, 2009] and DFKI's AILA [Lemburg et al., 2011]. Other well-known humanoids are still not torque controlled at all as, e.g., the legged humanoids Honda Asimo [Sakagami et al., 2002] or HRP-4 [Kaneko et al., 2011].

Hands and Tactile Sensing

The fingers of Agile Justin's hands have three actuated joints, like the human fingers (except for the thumb), and torque sensors in each joint. Some of the other advanced humanoid robots (the Armar family, iCub, TWENDY-ONE and AILA) have quite sophisticated hands but, except for the Sandia Hand [Labs, 2012], they are all underactuated with a maximum of two actuated DOF per finger, which limits their fine manipulation capabilities.

Some of these advanced hands are also equipped with high-resolution tactile sensors, but for none of them it has been shown that they can perform such a highly sensitive task like identifying an object's material just by touching it. As we show in this thesis, Agile Justin can perform this tactile material classification task with even superhuman performance, although it uses only a commercially available flexible tactile skin that simply has been taped onto its soft fingertips. The key for Agile Justin's highly developed tactile sensing lies in the advanced end-to-end deep learning methods used for interpreting the resulting complex and noisy spatio-temporal signal. Classical learning methods with manually designed features perform only in poorly on these skin signals.

Robotic Software Frameworks

Many of the most performant robotic systems in the challenging field of mobile manipulation bring their own software framework, e.g., iCub with YARP [Metta et al., 2006], PR2 [Willow Garage] with ROS [Quigley et al., 2009], AILA with ROCK [Rock], which is based on Orocos [OROCOS], in combination with ROS, or the recent Armar-6 with ArmarX [Vahrenkamp et al., 2015].

To cope with the complexity of advanced mobile manipulation tasks with their deep sensor-perception-planning-action loops running on distributed sensor, actor and computing resources, all robotic software frameworks follow a similar component based system architecture. A component implements a well defined part of the robot's functionality and communicates by means of a packet oriented transport layer with other components, be it on the same host or distributed in the network. But for complex dynamic manipulation tasks, such a component based architecture has to be supported despite tight timing constraints.

As we will show in this thesis, Agile Justin's software framework aRDx is currently the only framework that provides the necessary hard realtime determinism and highly performant data transport. E.g., Orocos claims realtime determinism but it is not performant, especially for large data packets. Other robotic frameworks have only recently started to work on achieving realtime performance, e.g. ROS 2 [Gerkey, 2015].

1.5 Outline and Contributions

The overall goal of this thesis is to extend the capabilities of the mobile torque controlled humanoid robot Agile Justin towards complex dynamic manipulation and highly developed tactile sensing to come closer to human versatility. This way, Agile Justin should become an almost ideal platform for research in intelligent mobile manipulation, especially in autonomous learning.

This *thesis by publication* is based on the papers attached in Appendix **??** and the *List of Publications by the Author* (cf. pp. 89) lists for each paper my share in percent as well as a short description of the particular contribution. We summarize the key contributions of this work in what follows and discuss them in more detail in the individual chapters, including some updated results and additional descriptions beyond the original publications. Each chapter ends with a short summary of its contributions, including the references to the corresponding publications.

Chapter 2 presents the realtime deterministic and highly performant communication layer of our aRDx software framework. First, we discuss the design considerations for a robotic software framework that can support the component based development of complex dynamic manipulation applications with their tight timing constraints. Then we present the elegant hierarchical implementation of aRDx which fulfills all design requirements by providing optimal data packet transport with zero-copy for intra-process and inter-process and copy-once for distributed communication, detailed control of the quality of service (QoS) and a powerful channel synchronization mechanism. When run on the realtime OS QNX, aRDx also achieves hard realtime determinism. Finally, in an elaborate stress test benchmark, we compare the communication performance of aRDx and aRD with prominent robotic software frameworks, namely ROS, Orocos and YARP. aRDx and its predecessor aRD outperform all other frameworks.

Chapter 3 describes the dynamic application of catching up to two thrown balls with Agile Justin hands which takes full advantage of aRDx's performance and advanced features. We give a brief overview of the challenging visual ball tracking using only onboard sensing while "everything is moving" as well as of the automatic and self-contained calibration procedure for the multi-sensorial upper body to provide the necessary precision. A major contribution is the unified generation of the reaching motion for the arms. The catch point selection, motion planning and the joint interpolation steps are subsumed in one nonlinear constrained optimization problem which is solved in realtime and allows for the realization of different catch behaviors.

Chapter 4 presents the highly sensitive task of tactile material classification with a flexible pressure-sensitive skin on Agile Justin's fingertip. We introduce our deep convolutional network architecture TactNet-II which directly works on the raw 16000 dimensional complex and noisy spatio-temporal tactile signal generated when Agile Justin sweeps with a finger over a material. All experiments are based on a new and large tactile dataset (3600 samples) with 36 typical household materials we recorded and which is made publicly available. We also perform for the first time a thorough human performance experiment with 15 subjects in which the human tactile performance is compared to the performance of a robot and which uses the very same 36 materials for both. The results we report show that Agile Justin reaches *superhuman* performance in the high-level or cognitive material classification as well as in the low-level material differentiation task. Finally, we adapt state of the art deep end-to-end n-shot transfer learning methods to our TactNet-II network architecture. The evaluation using our challenging 36 material dataset shows for the first time that deep end-to-end learning is feasible for the real world task of tactile material classification. Due to the knowledge transfer from a previously learned material classification task, an up to 15 fold reduction in the number of training samples required could be achieved for a new classification task with new materials.

Chapter 5 concludes this thesis by summarizing the presented work and giving an outlook of the usage of the upgraded humanoid Agile Justin.

In the Appendix A we provide a list of videos of the presented work and their weblinks for easy access.

Chapter 2

The Communication Layer of the aRDx Software Framework

2.1 Motivation and Related Work

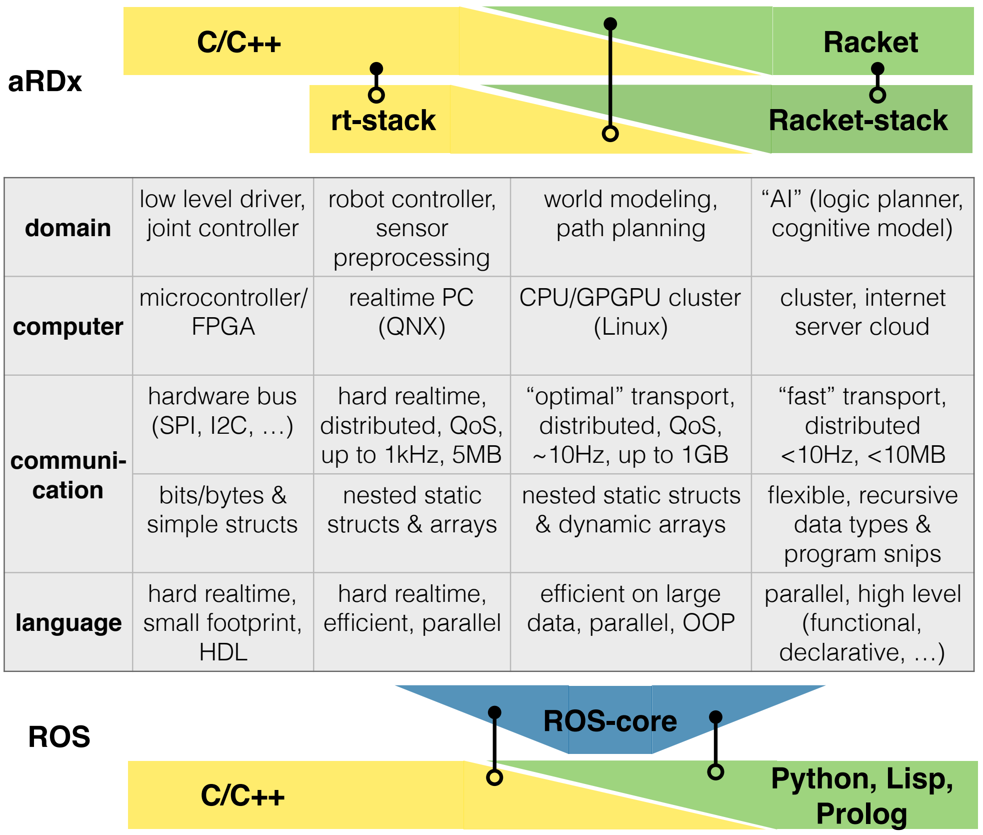
Advances in the performance of robotic systems are driven by the co-development of robotic hardware and software, with the software part becoming more and more important in recent years. This holds especially in the challenging field of mobile manipulation, where many of the most performant robotic systems bring their own software framework.

It is not surprising that the robotics community developed its own software frameworks and could not apply existing software concepts from other domains. The challenges in robotics are unique in their combination of a complex system architecture with, e.g., distributed sensors, actuators and computing resources, and the necessity that a robotic application has to span all abstraction levels, ranging, e.g., from hardware drivers over realtime motor controllers to whole-body motion planning and symbolic task level intelligence. Fig. 2.1 gives an overview of the different domains needed in an advanced robotic system.

To cope with these challenges, all robotic software frameworks follow a similar component based system architecture. Firstly, a component implements a well defined part of the robot's functionality and, secondly, it is a distinct execution entity (often an OS process) which communicates by means of a packet oriented communication layer with other components. Important benefits of such a component based approach are,

- robustness due to process boundaries with, e.g., memory protection,
- concurrent, parallel and even distributed execution of components,
- a decoupled development flow for a team of experts working on the diverse functionalities of a robotic application.

As already mentioned in Sec. 1.4, the software frameworks used in the most advanced humanoid robots are YARP [Metta et al., 2006] on iCub, ROCK [Rock], which is based on Orocos [OROCOS], in combination with ROS [Quigley et al., 2009] on Aila, and ArmarX [Vahrenkamp et al., 2015], which is based on the ICE middleware [Henning, 2004], on the recent Armar-6. ROS, the most widely used software framework in modern robotics and especially in the challenging field of mobile manipulation, also plays a growing role in humanoid robotics.



domain	low level driver, joint controller	robot controller, sensor preprocessing	world modeling, path planning	"AI" (logic planner, cognitive model)
computer	microcontroller/ FPGA	realtime PC (QNX)	CPU/GPGPU cluster (Linux)	cluster, internet server cloud
communi-cation	hardware bus (SPI, I2C, ...)	hard realtime, distributed, QoS, up to 1kHz, 5MB	"optimal" transport, distributed, QoS, ~10Hz, up to 1GB	"fast" transport, distributed <10Hz, <10MB
	bits/bytes & simple structs	nested static structs & arrays	nested static structs & dynamic arrays	flexible, recursive data types & program snips
language	hard realtime, small footprint, HDL	hard realtime, efficient, parallel	efficient on large data, parallel, OOP	parallel, high level (functional, declarative, ...)

Figure 2.1: The software domains of an advanced humanoid robotic system like Agile Justin, ranging from device drivers up to artificial intelligence. For each domain, the computational resources and operating system it is typically implemented on, the communication requirements, the complexity of the data structures that have to be transported, and the requirements for the used programming languages are listed. In addition, it is depicted for what domains the ROS (below) and aRDx (upper) frameworks can support a component based approach, what programming languages are used and how they are coupled with the communication layer.

In the following, for each of the two extreme ends of the range of software domains in an advanced robot system, the requirements a software framework must meet to support a component based approach for them are discussed.

2.1.1 Raw Communication Performance

All the above mentioned software frameworks have been successfully used in advanced applications. However, when it comes to take full advantage of the hardware capabilities of robotic systems like Agile Justin, a limitation becomes apparent. When building applications for such systems, not only the abstractions offered by the component based approach are important to cope with the systems' complexity, but also the raw performance of the framework's communication layer becomes essential. Here we mean communication performance in two extreme categories:

- **Latency** or realtime determinism for small (about 1 KB) data packets, as is usual for the low-level communication in motor control running in the kHz range and, hence, demanding for a jitter $< 100\,\mu$s. This corresponds to the second domain in Fig. 2.1.

- **High bandwidth** for large (>1 MB) packets, as is typical for image and skin data (e.g., a 2 megapixel stereo-camera system at 25 Hz generates 100 MB/s, or a skin with 3000 taxels at 750 Hz results in 2.25 MB/s). This corresponds to the second domain in Fig. 2.1.

We found out about these limitations with respect to raw communication performance by performing elaborate stress-test benchmarks for the most popular frameworks in humanoid robotics, i.e., ROS and YARP. We also included Orocos as a framework specifically dedicated to realtime applications, but found that even this framework does not achieve the necessary performance, especially for large data packets.

The results of this in-depth performance comparison are reported and discussed in Sec. 2.4. In difference to the comparison of the communication performance of robotic frameworks in [Einhorn et al., 2012], our benchmarks cover far more of the important aspects, e.g., scaling with the number of clients and distributed communication. Moreover, due to the "stress" character of our tests, we could uncover a number of severe quirks for many of the frameworks, which [Einhorn et al., 2012] did not see.

The found performance limitations of the frameworks can not be explained by "physical" limits of the underlying hardware, because modern computing resources with multicore CPUs and clocks in the gigahertz range as well as network interfaces with 1 GB/s transport should readily be able to provide the necessary computing power and communication bandwidths. Therefore, we decided to develop a new software communication layer with its main focus on performance. The result of this effort is the highly performant and realtime capable communication layer of our robotic software framework aRDx (agile robot development – next generation).

aRDx has a predecessor, our aRD (agile robot development)[1] framework [1], which was co-developed with the first version of the then stationary humanoid upper body Justin [Ott et al., 2006] out of the need to support the execution of complex and computationally intensive control algorithms on distributed resources. Therefore, aRD provides a tight coupling to MathWorks' Simulink and Realtime Coder [MathWorks] and it is also hard realtime capable, but only supports point-to-point connections between components and is only performant for small data packet sizes.

2.1.2 High-Level Domain Data Types

But also for the highest level domain, i.e., the AI or cognitive domain (see last column in Fig. 2.1), the above described frameworks are no ideal fit, as the packet data types they can transport between components are too restrictive. Typically, these frameworks only support nested static structs and one-dimensional dynamic arrays as, e.g., in ROS. But what would be needed in this highest level domain are flexible, recursive data types like trees and graphs.

A key idea in the design of the aRDx framework is that a single communication stack can not fulfill all demands of the wide range of domains in a robotic application, but that there have to be two stacks. The more static but highly performant and hard realtime deterministic stack, which we present in this thesis, and a flexible but less deterministic high-level stack. In aRDx, the high-level stack and all other higher level functionalities and abstractions needed in a robotic framework are implemented in a modern high-level programming language of the Scheme/Lisp family [20], i.e., Racket [Racket]. For example, we directly use Racket's in-built support for the serialization of arbitrary data structures or its advanced synchronization support for channel based communication.

Fig. 2.1 sketches, for which range of domains of an advanced robotic application aRDx or ROS can provide a component based approach. In aRDx's communication layer, the realtime stack (rt-stack) reaches down to the robot controller domain and the high-level Racket-stack up to the AI domain. ROS, on the other hand, can only reach slightly below and above the modeling and planning domain, as it neither supports realtime nor complex data structures.

In this thesis, we focus on aRDx's realtime stack of its communication layer. For simplicity, we will often use the term aRDx in the following, even if we only mean this realtime communication stack.

2.1.3 Recent Developments

Since about 2015, also other robotic software frameworks started to add realtime capabilities to fulfill the demands of advanced robotic systems, hence, following the path aRD and aRDx laid out already in 2006 [Bäuml and Hirzinger, 2006].

In [Paikan et al., 2015], YARP has been extended with run-time channel prioritization to increase the determinism and performance of packet transport under load. It uses a

[1]The name "aRD" (Agile Robot Development) was inspired by three points: (1) the concept allows for the realization of "agile robots", that is "fast, reactive and intelligent" robots demanding fast control rates and high computational power; (2) it supports an agile development flow for robotic systems; and (3) it was itself developed in an agile process during the work on our humanoid robot Justin.

similar approach to aRDx, in that it also only relies on services provided by the operating system, especially the scheduling priority for the threads implementing the packet transport. In addition, it uses packet quality of service of the operating system's network stack. Although the performance could be increased, YARP still does not reach hard realtime determinism, as the developers state themselves in [NATALE et al., 2016]: "For applications that require lower latency and higher determinism Orocos and aRDx may be a preferable choice."

In 2015, the OSRF (Open Source Robotics Foundation) released the first alpha version [OSRF, 2015] of a complete redesign and re-implementation of ROS, named ROS 2 [ROS2]. One of the design goals for ROS 2 was the support for hard realtime determinism [GERKEY, 2015] [KAY, 2016]. Other than ROS 1, ROS 2 does not implement the actual packet transport layer itself but it is built on top of DDS (Data Distribution Service) [PARDO-CASTELLOTE, 2003], a middleware standard for realtime systems by the Object Management Group (OMG). There are a number of open source and commercial vendors of DDS implementation and the goal for ROS 2 is to support many different vendors. Currently, eProsima's FastRTPS [EPROSIMA] as default, RTI's Connext [RTI] and ADLINK's OpenSplice [ADLINK] are supported.

But even in the most recent ROS 2 release (Bouncy Bolson, mid 2018) [OSRF, 2018a] the support for realtime is still only rudimentary. There is no realtime-safe intra-process messaging. This is only planned for future releases [OSRF, 2018b].

Preliminary benchmarking results [GUTIÉRREZ et al., 2018] on a simple distributed setup with only two components show that even to achieve only soft realtime performance, the threads of the underlying DDS implementation have to be manually tweaked specifically for the actually used DDS implementation. That means the abstraction ROS 2 wants to provide breaks down.

In a recent contribution to the *ROS Discourse* discussion group, Dejan Pangercic, a member of the ROS 2 Technical Steering Committee [GERKEY, 2018], lists a number of points that are still missing in ROS 2 to make it realtime capable [PANGERCIC, 2018]:

- Only the Connext Micro DDS [CONNEXT, 2018] implementation is hard realtime capable, but ROS 2 is only planning to support this DDS implementation in the future [OSRF, 2018b].
- The C++ client library, rclcpp, providing the communication API to ROS 2 applications, needs to be memory audited. E.g., if there are STL containers used, a realtime memory allocator would have to be provided.
- Memory allocators for realtime need to be passed correctly between the ROS 2 layers.
- All threads in ROS 2 need to have controllable stack sizes and priorities.
- The standard GCC exceptions which dynamically allocate memory have to replaced by static C++ exceptions.

Although the packet transport in ROS 2 is more deterministic than in ROS 1, [MARUYAMA et al., 2016] report that the average latency is worse than in ROS 1. This holds for all benchmarked DDS implementations and their analysis shows that the reason are the additional conversions of the data packets between the ROS 2 and DDS packet representation.

In contrast to ROS 2, the core of aRDx's performant communication stack directly sits on top of the operating system functionality without any additional third party software layers. We think this design decision is one important reason for aRDx's high performance and realtime determinism.

2.2 Design Considerations

The most important aspect for the design of aRDx's communication layer is the desire to reach the best performance possible with respect to latency and bandwidth, coming as close as possible to the limits of the underlying hardware (CPU, network, ...) and operating system. Other sources for the design decisions are the experience from working for many years with our former robotic software framework aRD on complex robotic systems, the inspiration from modern programming languages and the insights from intensively studying the strength and weaknesses of other robotic frameworks.

In what follows, we list and shortly discuss the important features and design decisions for aRDx.

- Packet based communication over abstract *channels* with many-to-many semantics (similar to ROS topics).
- Each channel is identified by a unique *channel-id* which can itself be sent over the channel.
- Flexible dynamic connecting and disconnecting to channels; each connection results in a *port*, being either a *put*- or a *get-port*.
- Channels transparently transport the packets over process and host boundaries. That means, the channel API is the same for the *process* = intra-process, *host* = inter-process and *distributed* = inter-host domain.
- Optimal transport in each communication domain:
  - *zero-copy* semantics in process and host domain,
  - *copy-once* to each host with ports connected to a given channel (in contrast to copy-once to each port on each host in a peer-to-peer model, such as, e.g., ROS and Orocos use).
- Detailed control of the *quality of service* by optionally specifying a *communication priority* for each connected port.
- *Hard realtime* determinism of the underlying OS is retained (e.g., for QNX, as we have shown earlier with aRDx's predecessor aRD, hard realtime performance can even be reached for distributed communication).

- To achieve realtime determinism, all resources of a given channel have to be *static*, hence, have to be determined when the channel is created (e.g., maximum packet size, maximum ring-buffer size, maximum number of ports, ...). All dynamic memory allocation would deteriorate determinism.

- No explicit serialization step with parsing/unpacking as this would reduce performance due to, in general, at least one additional copying of the data.

- *Time-order* of packets sent from the same host is kept intact, only the relative order of packets sent from different hosts is not guaranteed. Otherwise, additional and costly inter-host synchronization for each packet would be necessary.

- Efficient *synchronization* mechanism inspired by Racket [RACKET] and Concurrent ML [REPPY, 1999] which allows to wait blocking on an arbitrary number of ports. This feature is of great value as it can often drastically reduce an application's complexity, which has to wait for data from different sources. E.g., in frameworks like aRD or YARP, this could only be solved by adding threads to the application, or for frameworks with purely callback semantics like ROS, additional states and control logic would have to be added.

- No model of computation should be enforced. A communication layer should transport data and be compatible with any model of computation which is optimal for a given application. This is possible when providing the sync mechanism of the last point, but not, e.g., for a callback model like in ROS/ROS 2.

- No additional threads should be started in the client making it easy for the application programmer to set the desired priorities in his application without having to deal with threads not under his direct control; this is in contrast to the *thread clutter* of other frameworks (e.g., YARP adds 2 threads for each connection to each port).

- On the client side, only POSIX [GALLMEISTER, 1995] primitives should be used, e.g., named shared memory, mutex and condition variables and TCP sockets. This usually guarantees best performance and introduces the least possible library dependencies and conflicts when, e.g., the communication layer should be linked to an already complex application (e.g., a Matlab/Simulink [MATHWORKS] model).

- Robustness against clients running amok, i.e., only the data of the channels this client is connected to with a put-port could be compromised, but not the overall channel logic and the rest of the communication net.

- Minimal and easy to use API. This is important as typical users are experts in robotics but not necessarily software experts and are not willing to invest much time to understand sophisticated software frameworks.

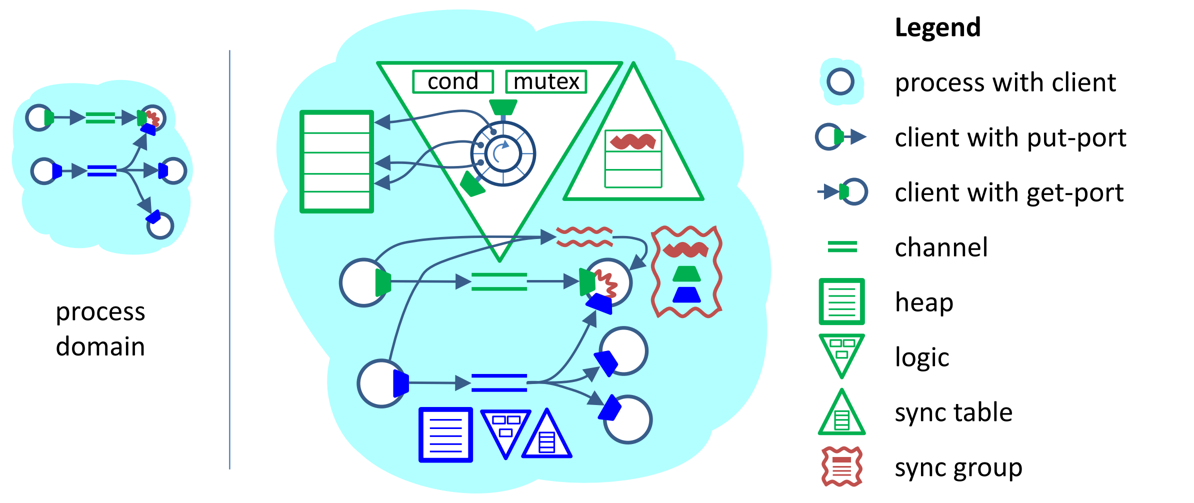


Figure 2.2: Process domain. In the abstract application view (left), all clients are running in the same process. aRDx can implement (right) this by directly using the basic channel consisting of a ring-buffer logic, simple POSIX synching data structures (a mutex and a condition variable) and a heap for the actual packets. This is depicted in detail for the green channel.

2.3 Implementation

The implementation of aRDx can fulfill all the design requirements including the zero-copy semantics for the process and host domain and optimal network transport for the distributed domain with a minimalistic and elegant hierarchical approach.

2.3.1 Data Packet Transport

The implementation for the process, host and distributed domains build on each other, starting with a simple basic channel.

To discuss the details of the implementation, we introduce a small example application (see Fig. 2.2, left) with five clients communicating with each other over two channels. The application includes a one-to-one and a one-to-many pattern and a sync-group for waiting blocking on two ports connected to each one of the channels. The Figures 2.2, 2.3 and 2.4 show for the process, host and distributed domain how a developer of such an application might map the clients to actual processes and computer hosts. The right part of each figure then shows for each of the domains how aRDx actually implements this abstract application's view.

Process Domain (Fig. 2.2)

When a client connects to a channel, it gets a port (depicted as a quadrangle with orientation, depending on wether it is a put- or a get-port). If a client wants to send a packet, it requests for a free slot in the channel's heap, writes in its data, and puts the packet into the channel by incrementing the put-head of the channel's ring-buffer (the put-head

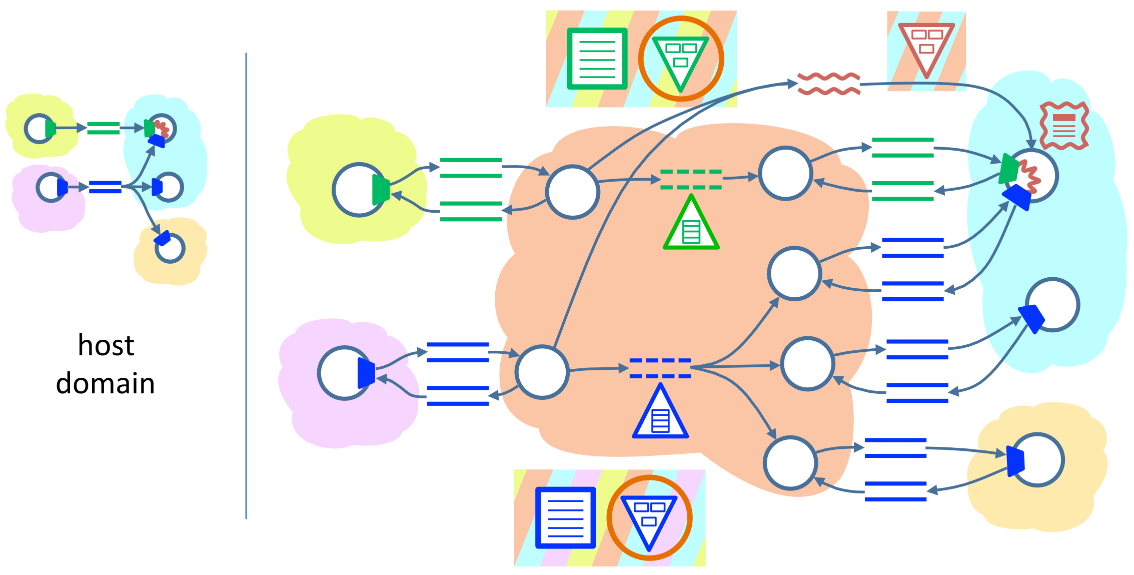


Figure 2.3: Host domain. In the application view (left) the clients are now mapped to four processes (clouds with different colors).

is depicted with the same quadrangle orientation a put-port has). Moreover it fires the condition variable signaling that new data is available.

If a client wants to do a blocking receive for a packet through one of its get-ports, it first checks if a packet, which this get-port did not get yet, is available from the (corresponding) channel's ring-buffer by comparing the get-port's head into the channel's ring-buffer with the channel's put-head. If there is a new packet, the index to its slot in the heap is returned and the slot is marked as "in-use", avoiding that it is given to a put-port for modifying it. After the data is processed, the client gives back the slot. If there was no packet available, the client waits blocking on the condition variable.

All of the "higher level" constructs, like sync-groups with the capability to wait blocking on more than one get-port (usually from different channels), are built from this basic channel mechanism. An example of a sync-group is depicted for the client in the upper right of the figure. A sync-group consists of a stripped down basic channel, only consisting of the mutex and condition variable and a list of the get-ports which are in the group. When a client blocks on a sync-group, it actually blocks on the corresponding basic channel. A sync-group gets fired whenever a packet is put in any of its get-ports' channels. This is done by the client when putting the packet into a channel by not only firing the channel's condition variable, but also running through a list stored in the channel with put-ports to all the sync-groups (resp. the underlying basic channels) and fires them too.

Host Domain (Fig. 2.3)

To be able to provide a zero-copy semantics also for the host domain, processes have to share memory to some extent. But aRDx obeys the process boundaries in all relevant aspects by introducing an additional daemon process (red cloud) that acts like an OS kernel regarding the communication for the clients. Only the daemon shares memory with all processes, but the memory protection between application processes which do

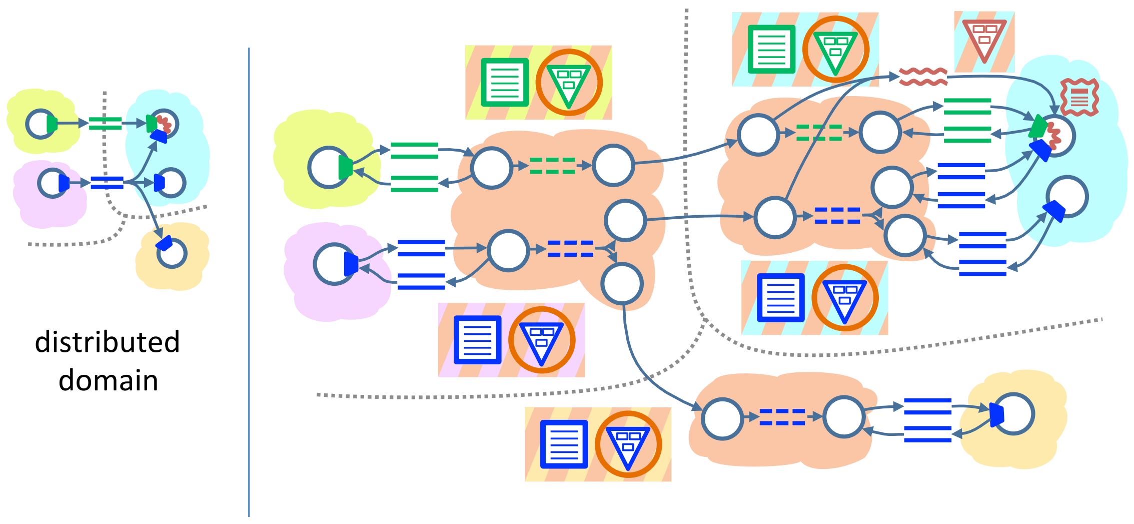


Figure 2.4: Distributed domain. The five clients of the application (left) are now running on three different hosts.

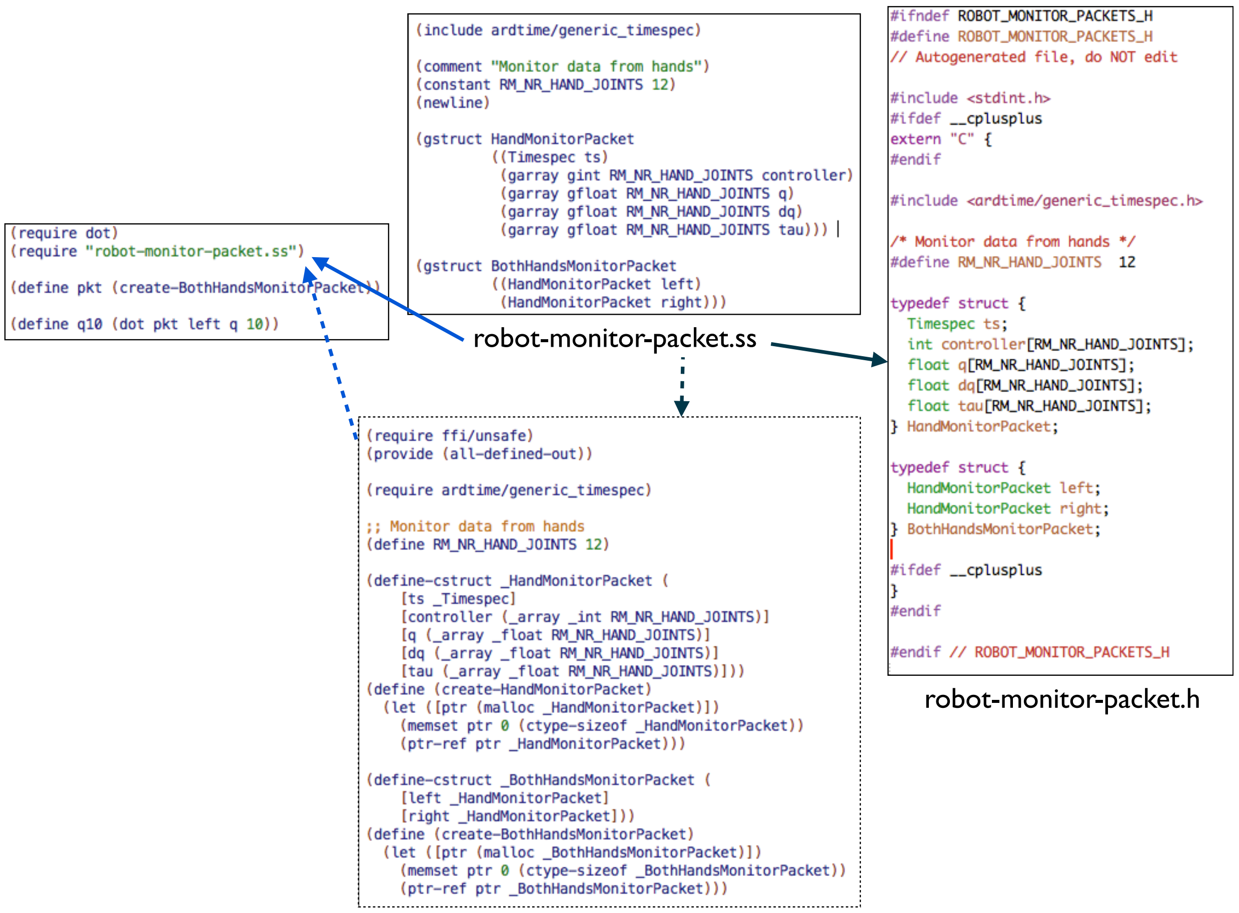
not communicate with each other is kept intact. But even two communicating application processes share only the actual data (in the heap), while the communication logic can only be modified by the daemon. This way, even if a process runs amok, the integrity of the rest of the communication net can not be violated.

Technically this is achieved by a ”mirror” thread in the daemon for each port in an application process. This thread is responsible for modifying the communication logic, when demanded by the port. This way all communication mechanisms from the process domain, including the sync-groups, can be directly reused without change. Interestingly, even the communication between the daemon’s threads and the port in the process is implemented by the very same basic channels that have been implemented for the process domain, but only mapped to shared memory.

Distributed Domain (Fig. 2.4)

Distributed communication crossing the host boundary is implemented with a similar idea as crossing the process boundaries in case of the host domain. A channel is mirrored on all hosts where at least one port has been connected to it. This mirroring is done in a performance optimal way by copying the data only once to the host – independent of the number of ports which are connected to the channel on the host. This can dramatically reduce the network bandwidth required compared to a simple peer-to-peer communication approach.

Technically, each host runs an aRDx daemon as described for the host domain and adds one additional thread for each mirrored channel and transfer direction, which sends or receives the packets over TCP/IP to/from the corresponding mirror channels on the remote hosts.



```
(require dot)
(require "robot-monitor-packet.ss")

(define pkt (create-BothHandsMonitorPacket))

(define q10 (dot pkt left q 10))
```

```
(include ardtime/generic_timespec)

(comment "Monitor data from hands")
(constant RM_NR_HAND_JOINTS 12)
(newline)

(gstruct HandMonitorPacket
         ((Timespec ts)
          (garray gint RM_NR_HAND_JOINTS controller)
          (garray gfloat RM_NR_HAND_JOINTS q)
          (garray gfloat RM_NR_HAND_JOINTS dq)
          (garray gfloat RM_NR_HAND_JOINTS tau))) |

(gstruct BothHandsMonitorPacket
         ((HandMonitorPacket left)
          (HandMonitorPacket right)))
```

```
(require ffi/unsafe)
(provide (all-defined-out))

(require ardtime/generic_timespec)

;; Monitor data from hands
(define RM_NR_HAND_JOINTS 12)

(define-cstruct _HandMonitorPacket (
    [ts _Timespec]
    [controller (_array _int RM_NR_HAND_JOINTS)]
    [q (_array _float RM_NR_HAND_JOINTS)]
    [dq (_array _float RM_NR_HAND_JOINTS)]
    [tau (_array _float RM_NR_HAND_JOINTS)]))
(define (create-HandMonitorPacket)
  (let ([ptr (malloc _HandMonitorPacket)])
    (memset ptr 0 (ctype-sizeof _HandMonitorPacket))
    (ptr-ref ptr _HandMonitorPacket)))

(define-cstruct _BothHandsMonitorPacket (
    [left _HandMonitorPacket]
    [right _HandMonitorPacket]))
(define (create-BothHandsMonitorPacket)
  (let ([ptr (malloc _BothHandsMonitorPacket)])
    (memset ptr 0 (ctype-sizeof _BothHandsMonitorPacket))
    (ptr-ref ptr _BothHandsMonitorPacket)))
```

```
#ifndef ROBOT_MONITOR_PACKETS_H
#define ROBOT_MONITOR_PACKETS_H
// Autogenerated file, do NOT edit

#include <stdint.h>
#ifdef __cplusplus
extern "C" {
#endif

#include <ardtime/generic_timespec.h>

/* Monitor data from hands */
#define RM_NR_HAND_JOINTS  12

typedef struct {
  Timespec ts;
  int controller[RM_NR_HAND_JOINTS];
  float q[RM_NR_HAND_JOINTS];
  float dq[RM_NR_HAND_JOINTS];
  float tau[RM_NR_HAND_JOINTS];
} HandMonitorPacket;

typedef struct {
  HandMonitorPacket left;
  HandMonitorPacket right;
} BothHandsMonitorPacket;

#ifdef __cplusplus
}
#endif

#endif // ROBOT_MONITOR_PACKETS_H
```

Figure 2.5: Example usage of aRDx's data packet description domain specific language. The file `robot-monitor-packets.ss` defines two new packet types which use the `Timespec` struct as one of their fields. The `Timespec` struct is defined in the file `ardtime/generic_timespec.ss` which is included by `robot-monitor-packets.ss`. In a Racket program (left), the `robot-monitor-packets.ss` file can be directly used as a module. It is compiled on the fly into the corresponding Racket bindings (middle, lower). It can also be converted into the corresponding header file (right) to use the packet type from C/C++.

2.3.2 Data Packet Serialization

To allow for efficient packet transport and zero-copy semantics, aRDx uses no explicit packet serialization step with parsing/unpacking. This is in accordance with the modern, highly efficient serialization protocols like Capn' Proto [CAP'N PROTO] or FlattBuffers [FLATBUFFERS] which use an encoding that is appropriate both as a data interchange format and an in-memory representation. Key is that this platform independent encoding can be efficiently accessed by modern CPUs.

aRDx puts this to the extreme and directly uses the memory layout of the GNU GCC compiler [GNU] with some additional memory layout pragmas. For describing the packet data types, we developed an embedded domain specific language (DSL) implemented in Racket. The DSL allows the description of any static C data struct including multi-dimensional arrays (for comparison, ROS, e.g., only supports one-dimensional arrays) and nested structs. A data description file can be directly used as a Racket module where the corresponding Racket bindings are generated on the fly for accessing the data types via Racket's FFI (foreign function interface). For other programming languages, the type description files are converted to corresponding language files, e.g., header files for C/C++. An important feature of our Racket based implementation of the DSL is that it generates meaningful error messages at the abstraction of the DSL and not the underlying base language.

Fig. 2.5 shows an example of a hierarchical data packet description file and its usage.

2.4 Performance Comparison

In this section we analyze the communication performance of robotic software frameworks and especially the communication layer of our aRDx framework.

2.4.1 Stress Test Setup

For comparing the raw communication performance of aRDx, ROS, YARP, Orocos and aRD we ran the following stress test: a master sends as fast as possible the same data of size P to a number C of clients (one-to-many), which, on reception, immediately send back a response packet of the same size P to the master (many-to-one). After the master has received all response packets it immediately, without any pause (hence, it is a stress test), starts a new round with sending again a packet to the clients and so on. The test is run for various packet sizes P and number of clients C and the round-trip time is measured at the master, i.e., the time one round of this ping-pong communication takes. We ran the tests for all the three domains, namely the process, host and distributed domains. For the distributed domain, three identical computer hosts coupled with switched 1 GigE were used. One of them was running the master and on the other two the clients were distributed equally, while increasing C. The transferred data was a simple byte array with size P, so that no complex serialization of the data was necessary as we are interested in the raw communication performance.

We did our best to implement this test as efficient as possible for all the frameworks by using all optimizations recommended in the respective documentation. We also tried to

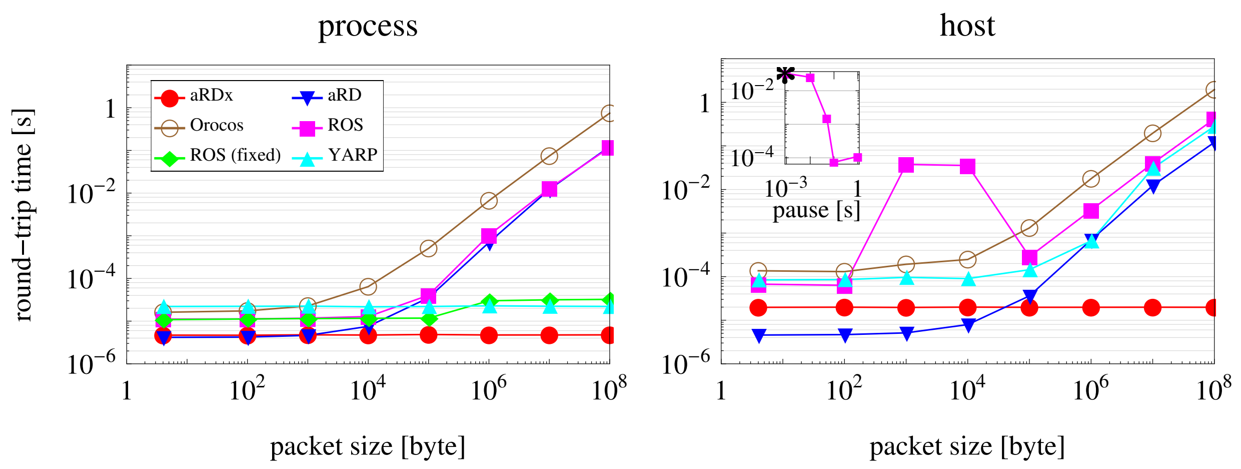


Figure 2.6: Results of the stress test benchmark for $C = 1$ client and for the process and host domains.

set realtime priorities for the master and the clients as far as it was possible due to the extreme thread-clutter of some frameworks. For aRDx, we also chose the most performant implementation variant with one one-to-many channel for transferring the packets from the master to all clients and one many-to-one channel for transferring the data back from all clients to the master. Other implementation variants with aRDx and their implications on performance are analyzed in section 2.4.2.

2.4.2 aRDx and Other Frameworks

For the performance comparison of aRDx with other popular frameworks, we ran the benchmarks on Linux. This OS is supported by all frameworks and, as it is the most widely used OS for advanced robotic systems, we expected the frameworks to be best optimized for Linux. The used computers were identical and configured as follows:
– Dell Precision T3500, 6 GB RAM,
– Intel Xeon W3530@2.80 GHz,
– 4 cores, hyper-threading turned off,
– SuSE Linux SLED11 SP2, kernel 3.0.58, 32bit (PAE), gcc 4.3.4.

Packet Size

Fig. 2.6 shows the results for $C = 1$ client and packet sizes from 4 to 100MB for the process and host domain. Each plot shows the mean round-trip time (averaged over some 100 runs) over the packet size for the various frameworks. Please be aware of the log-log-scaling of the plots.

The performance of aRDx is almost always the best – most dramatically for the host domain where no other framework can provide zero-copy semantics. Only for small packet sizes (up to 1 KB), where the transfer time is dominated by the constant overhead of a framework, aRDx is beaten by aRD's minimalistic implementation.

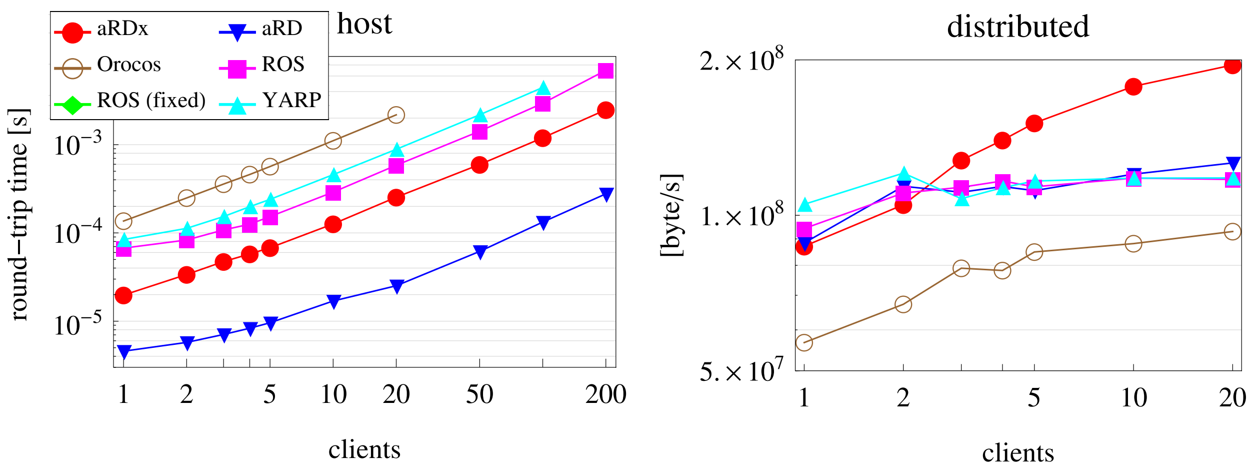


Figure 2.7: Benchmark results plotted over the number of clients to highlight the frameworks' behavior for the two extreme aspects of performance: *latency* (left, showing the round-trip time for 4 Byte packets) and *effective bandwidth* (right, showing the effective bandwidth for 1 MB packets).

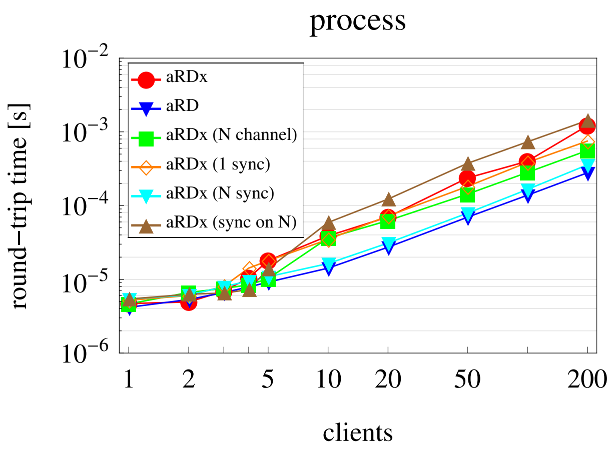
In what follows, we discuss some features and quirks of the other frameworks we came about. All these frameworks scale very well and roughly linear with the number of clients. For the process domain, YARP can provide zero-copy semantics. In this domain, ROS with its nodelets also was expected to show constant transfer times but could do so only after we fixed the implementation (labeled ROS fixed). Standard ROS (labeled ROS) completely initializes the memory of newly constructed packets, hence, the transfer time scales with the packet size.

For the host domain, YARP and ROS perform very similar, since both communicate over loopback TCP sockets. In the case of large packets ($>$ 1MB), they even reach almost the performance of the shared memory based transport of aRD, showing that the Linux loopback sockets are very efficient.

In all tests, the performance of Orocos was worst, although we always tried the optimal parameters. We suspect that this comes due to the additional abstraction layer with ACE/TAO in Orocos's communication stack. For ROS, we found another severe quirk in the host and distributed domain and packet sizes of 10KB to 100KB. There, the round-trip time dramatically increases 100x. A further analysis showed that this effect disappears completely when adding a pause of at least 100ms between each round of the test (see the inset in the 1-client plot depicting the round-trip time over the pause time for 1KB packet). This means, ROS is not really stress resistent.

Latency and Bandwidth

Fig. 2.7 discusses in more detail the two extreme aspects of performance over the number of communication clients: latency with 4 Byte packets for the host domain and bandwidth with 1 MB packets for the distributed domain. For the latter, the effective bandwidth b_{eff} is defined as the summed up number of bytes transferred in each round of the test between the master and the C clients divided by the round-trip time Δt: $b_{\text{eff}} = 2CP/\Delta t$.



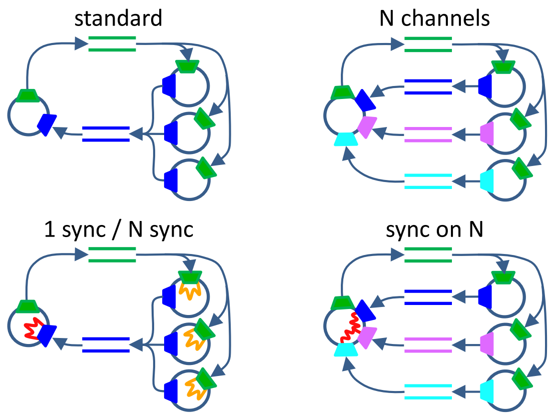


Figure 2.8: In-depth analysis of aRDx. The stress-test benchmark was implemented with different channel and sync-group configurations (right) and performance results for the 4 Byte packet size (hence, latency performance) are compared for the process domain (left).

As already discussed in Fig. 2.6 for the process and host domain, the minimalistic aRD has the smallest latencies for small packets, but aRDx still performs 4x better than all other frameworks. For all frameworks, the latency scales linearly with slope 1, showing that the interplay of the frameworks' internal logic and the Linux scheduler works nearly optimal, e.g., the scheduler can distribute the load on all 4 CPU cores.

For the distributed domain, all frameworks, except aRDx, have a peer-to-peer semantics, leading to a saturation of the effective bandwidth at the physical bandwidth limit of the GigE interface of about 110 MB/s. In contrast, the effective bandwidth of aRDx keeps increasing with the number of clients as it has to transfer the data from the master to the clients only once for each of the two hosts.

As a side note, again Orocos performed by far the worst, and we could not run the test beyond 20 clients without crashes. For YARP, we could not run the tests beyond 50 clients for the process and 100 clients for the host domain without crashes. Presumably this happens due to the thread-clutter YARP generates with 3 threads per port and 2 threads for each connection to a port.

2.4.3 In-Depth Analysis of aRDx

To get a deeper insight into the performance of aRDx, we analyzed how it scales with the number of channels and the impact of adding sync-groups. In addition, we ran tests on the realtime OS QNX to demonstrate aRDx's realtime determinism and compare the results to the Linux case.

Varying Configurations

The stress-test benchmark has been implemented with different numbers of channels and sync-groups. Fig. 2.8 shows the performance results for the process domain as the impact of the different configurations is the biggest for this domain.

For all depicted configurations, the master sends its packets to a single channel to which all clients are connected to.

- In the ″standard″ configuration, the clients send their response packets over one shared channel to which the master is connected to. The master then waits in a loop, blocking on its get-port for all clients' responses.
- In the ″1 sync″ configuration, the master has a sync-group containing the get-port, so that each client, in addition to putting its packet into the response channel, also has to fire the sync-group. The master now waits in its loop blocking on the sync-group, only subsequently getting the packet from the get-port.
- In the ″N sync″ configuration, each client has a sync-group with its respective get-port in it. Now the master has to fire all sync-groups every time it is putting a packet into the channel.
- In the ″N channel″ configuration each client has its own response channel, so that they can put in their packets in parallel. The master now waits blocking for one get-port after the other.
- In the ″sync on N″ configuration, in difference to the previous one, the master has one sync-group containing all its get-ports for the response channels and, instead of looping over the get-ports in fixed order, now blocks in its loop on the sync-group and subsequently gets the packet from the get-port which activated the sync-group. In this configuration, all clients have to fire the same sync-group, hence, the parallelism in sending the response packets is reduced.

The performance results show that even for the process domain, the different configurations have only minor influence on the latency – maximally a factor of 2 compared to the standard configuration. Only the difference in the parallelism possible between the ″N channel″ and ″sync on N″ configurations is clearly visible. For the host domain, the differences in the latencies relative to the standard configurations are even smaller and almost non-existent.

The results show that aRDx's implementation is very efficient and can easily handle hundreds of channels and sync-groups with almost no influence on the performance.

Hard Realtime

To prove the hard realtime determinism of aRDx, we ran the stress-test on the realtime OS QNX and measured, besides the average, also the worst-case performance. Fig. 2.9 shows the results and compares them to the performance for Linux.

The plots show the average and, depicted as an error-bar, the minimal and worst-case round-trip time. In the distributed domain, the error-bars are omitted for Linux because worst-case timing would be out of scale. When run on QNX, aRDx is highly deterministic with very little jitter ranging from about $10\,\mu$s (compared to $200\,\mu$s on Linux) for small client numbers and the process domain up to only 1 ms for 200 clients and the host domain (not plotted). This excellent realtime behavior is complemented with almost the

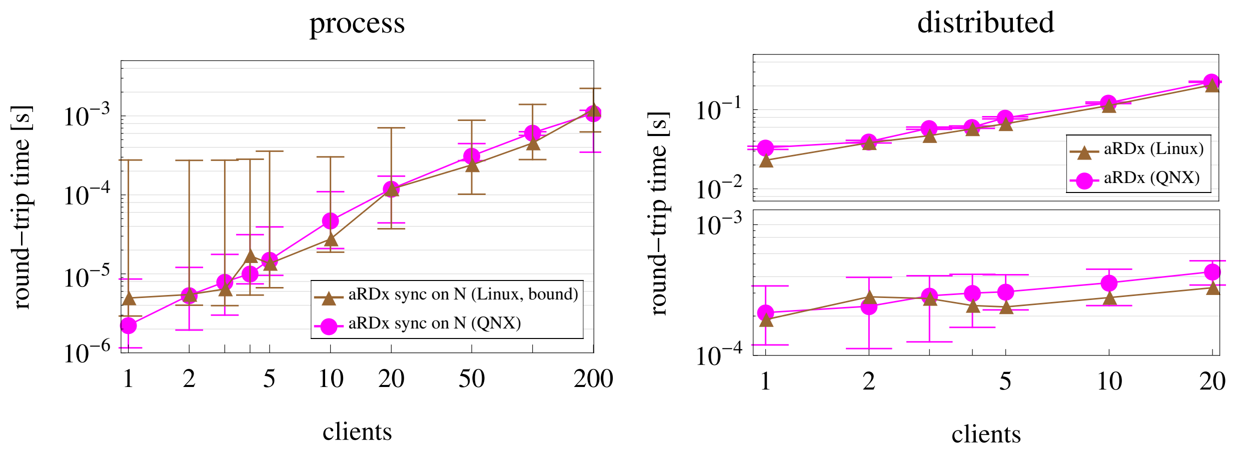


Figure 2.9: Realtime determinism of aRDx. For the process domain (left), all tests were run with 4 Byte packets (latency performance) and in the "sync on N" configuration (most challenging internal synchronization logic) on the realtime OS QNX and, for comparison, on Linux with the clients bound to the CPUs (best determinism on Linux). For the distributed domain (right), the tests were run for 4 Byte (lower) and 1 MB packets (upper).

same average performance as for Linux. Even for the distributed domain, the combination of aRDx and QNX's TCP/IP stack could reach a worst-case round-trip time of $< 500\,\mu$s. This proves that applications with hard realtime demands in the 1 kHz range can be implemented on distributed computing resources by aRDx/QNX. Finally, the performance of QNX's TCP/IP stack was tested for throughput by running the tests with 1 MB packet size. Even here, it performs very well compared to Linux.

In summary, the combination of QNX and aRDx reaches excellent hard realtime determinism for the host and process domains, easily allowing for applications with 100 clients running in the kHz range. Even for the distributed domain, the worst-case round-trip latencies are no larger than 500 μs.

2.5 Small Example and Real World Applications

In this section we first present a small example of the raw aRDx API and then discuss the system architecture of Agile Justin as a complex real world example. Finally, we show the broad range of robotic domains in which aRDx and aRD are used beyond humanoid robots.

2.5.1 Small aRDx Client

The low-level API of aRDx is a plain C-library without any dependencies on other frameworks or libraries except for the standard C (POSIX compatible) library. The API consists of management functions for creating and removing channel-ports as well as sync groups and only five main functions for handling packets. The code example in Fig. 2.10 outlines the usage of these functions.

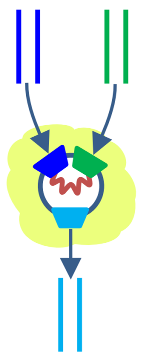
In this low-level interface, packet data may, by convention, only be modified by a sending client and never be read or written after the ownership is released. In a higher-level

```
 1 #include <ardx/ardx.h>
 2 #include "mypacket.h"
 3
 4 int main(int, char**) {
 5     ardx_channel_id_t pcid  = {42, ARDX_CHANNEL_DOMAIN_HOST},
 6                       gcid0 = {43, ARDX_CHANNEL_DOMAIN_HOST},
 7                       gcid1 = {44, ARDX_CHANNEL_DOMAIN_HOST};
 8     ardx_channel_port_t *pport, *gports[2];
 9
10     size_t pkt_size = sizeof(mypacket_t);
11     pport =  ardx_connect_channel(&pcid, ARDX_CHANNEL_MODE_PUT, pkt_size);
12     gports[0] = ardx_connect_channel(&gcid0, ARDX_CHANNEL_MODE_GET, pkt_size);
13     gports[1] = ardx_connect_channel(&gcid1, ARDX_CHANNEL_MODE_GET, pkt_size);
14
15     ardx_sync_group_t *sg = ardx_create_sync_group(gports, 2);
16
17     while (1) {
18         ardx_channel_port_t *port = ardx_sync(sg);
19         ardx_packet_t* gpacket = ardx_channel_get(port);
20         mypacket_t *gdata = (mypacket_t*)ardx_packet_data(gpacket);
21
22         ardx_packet_t* ppacket = ardx_malloc_packet(pport);
23         mypacket_t *pdata = (mypacket_t*)ardx_packet_data(ppacket);
24
25         // process data ...
26         pdata->value = gdata->value+1;
27
28         ardx_channel_put(pport, ppacket);
29         ardx_release_packet(port, gpacket);
30     }
31 }
```

(a)

```
typedef struct _mypacket_t {
    int value;
} mypacket_t;
```

(b)



(c)

Figure 2.10: The C-source-code in (a) shows a fully functional aRDx-client-program that works on structs of the type `mypacket_t` given in (b). As illustrated in the abstract view (c), the example program consists of a single process with a single thread. It can read data from two get-ports, both inside a sync-group, connected to two different channels. A single put-port allows to send data to a third channel. The program is composed of the the specification of the used channels (L5–7), followed by the connection of the three ports to these channels resulting in two get- and one put-port (L11–13). Both get-ports are combined in one sync-group (L15). The main processing loop first waits via sync on new data on one of both get-ports, then retrieves a handle to the packet and a pointer to the data area from the returned port (L18–20). After allocating a packet-handle for the put-port, together with a pointer to its data area (L22, 23), both data pointers are accessible. This allows to generate the outgoing data in-place (L26). Afterwards, the outgoing packet is sent and the incoming packet is released.

language wrapper, this is enforced automatically (e.g., in C++ using a kind of smart pointers for resource management) as well as type safety is guaranteed.

2.5.2 Mobile Humanoid Agile Justin

The development of the aRDx framework and its predecessor, the aRD framework, has been driven by the demands of the advanced mobile humanoid robot Agile Justin. Fig. 2.11 shows the implementation of Agile Justin's system architecture based on aRDx. The high performance and hard realtime capabilities of aRDx are key for the versatility of Agile Justin, allowing it to perform time-critical tasks like catching and throwing balls or to perform large scale data processing in the context of 3D-vision and dextrous fine manipulation [5].

The usage of the aRDx framework for Agile Justin has several important advantages. One is the described raw communication performance for high bandwidth as well as low latency settings. The high bandwidth allows to distribute large amounts of data, as they are generated from the cameras, Kinect and skin sensors efficiently between different processes and, when needed to the outside systems. With the help of the daemons running on each computer, it is guaranteed that always the least possible data is transferred. Likewise, aRDx's very low latency makes it possible to distribute even control loops in the kHz range over multiple processes and even hosts, as is done on the onboard QNX system.

Another advantage of the low latency communication, especially in the process and host domain via zero-copy, is that the aRDx's communication primitives can be easily used for intra-process communication in high performance scenarios. This greatly simplifies inter-thread communication and can significantly reduce concurrency related errors. If a process uses aRDx-channels for all internal communication, it could be easily split up into different processes, thus improving encapsulation, stability and reusability without any performance penalties. In the presented system, an example for the reusability are the multiply used image compression and decompression components. Before the aRDx based implementation, these components had to be part of the corresponding monolithic algorithmic processes for performance reasons.

The viewers and controls GUIs running on the user interface computer are also connected via aRDx channels. They allow to observe or control the sensor input and state of the system throughout the processing loop. This is only possible because additional receivers have the least possible influence on the rest of the system. They cannot block senders or other receivers directly or indirectly by additional copying of data in the process or host domain and they have, due to the copy-once design, the least possible influence in the distributed domain. Therefore, these user interface programs can be connected and disconnected at arbitrary points in time without danger of violating timing constraints, neither in the sensor-perception-planning-control loop nor in the hard realtime motor control loop.

A particularly impressive demonstrator which takes full usage of aRDx's performance is the ball catching scenario which we will present in more detail in Chap. 3.

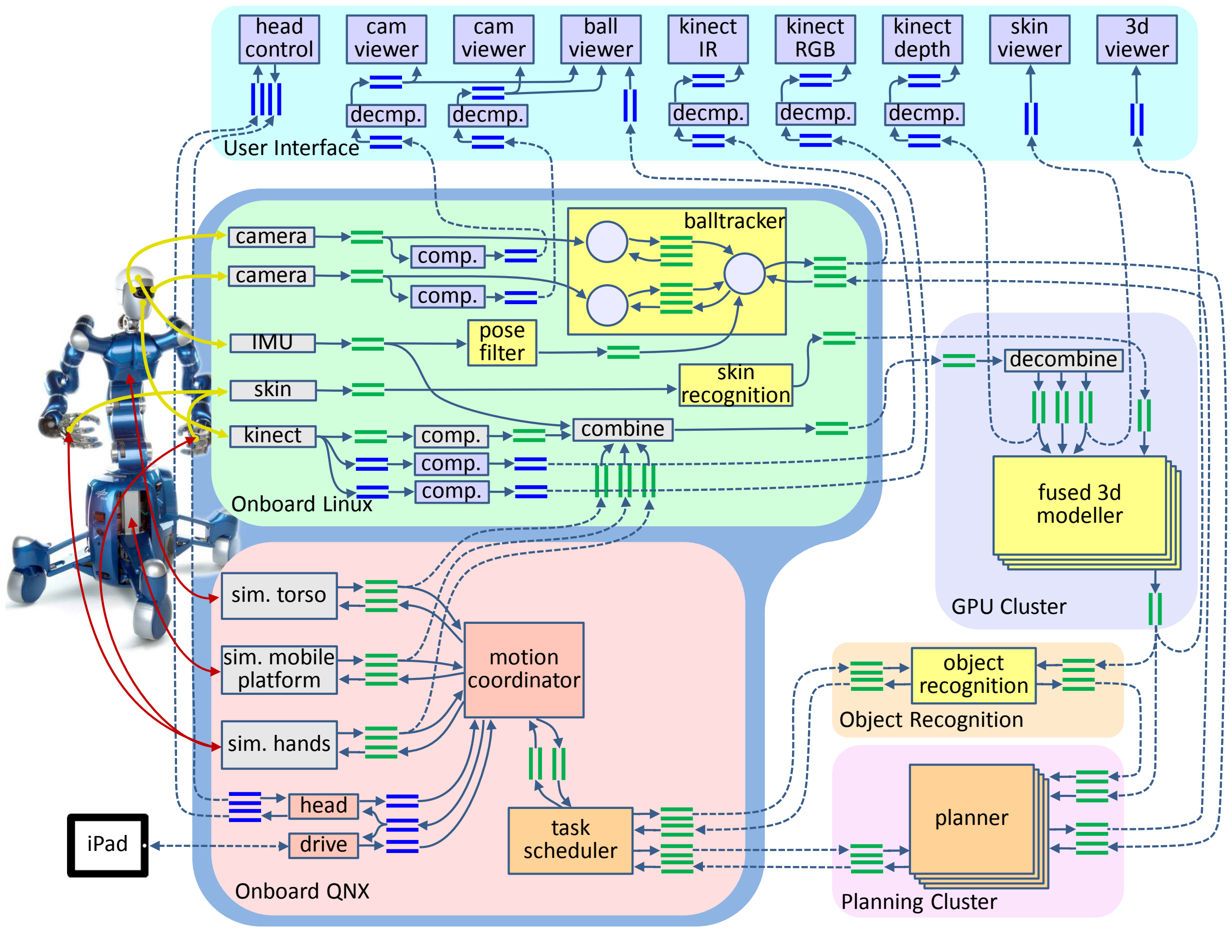


Figure 2.11: The communication architecture of Agile Justin. The computing resources consist of two onboard (Linux and QNX, blue framing) and multiple computers and clusters outside the robot (computers are marked as colored rounded boxes). Each of these two groups can internally communicate over GigE, but are connected with each other via WLAN (each rectangular block is a single process, mirror channels are connected by dashed arrows). The system architectures comprises one large sensor-perception-planning-control loop spanning all computing systems. This is in contrast to more classical action-perception loops which run on a single computer and do not contain higher level perception models or global planning algorithms. The sensory input is completely generated onboard, preprocessed there and then sent by WLAN outside for further processing. After higher level processing and planning on these informations, the resulting desired actions are sent back, where they are handled by the realtime QNX system, which runs the 1/,kHz control loop spanning multiple processes. To efficiently transfer the data over the low bandwidth WLAN connection, the data is compressed (comp. blocks) before transmission and decompressed afterwards (decmp. blocks). Another way to reduce the communication load, is to transfer the sensor input at the lowest needed rate and to transfer them only once outside the system, even when multiple receivers exist. This is, e.g., done for the skin data and Kinect depth images, which are needed for the 3D-modelling as well as the user interface (blue channels and boxes at the top).

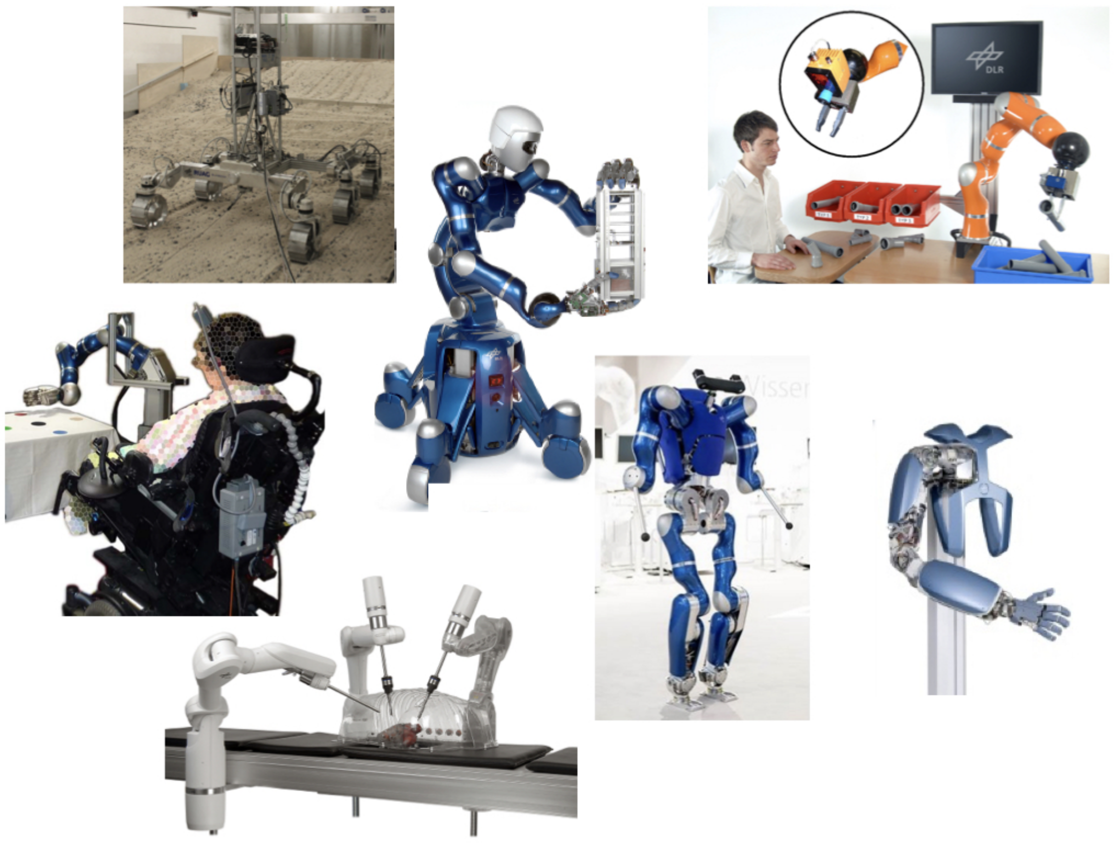


Figure 2.12: Robotic domains in which the aRD and aRDx frameworks are used. Clockwise, starting in the upper left: ExoMars Rover, Agile Justin, human robot cooperation (HRC), DLR antagonistic humanoid HASy, DLR bipedal humanoid Toro, MiroSurge, Brain-controlled assistent. (Pictures are taken from the corresponding publications as referenced in Sec. 2.5.3.)

2.5.3 Other Robotic Domains

Although originally developed for the advanced humanoid Agile Justin, the aRDx and aRD frameworks have proven valuable in a wide range of other robotic domains (see also Fig. 2.12). Especially aRD allows due to its very lightweight implementation for easy and non-invasive integration in existing software tools and porting to new OSes. E.g., besides Linux and QNX, which aRDx supports, aRD has been ported to VxWorks, Windows and iOS as well.

- *Humanoid Robots*: The development of aRD started with Agile Justin's predecessors, the stationary humanoid upper body Justin [Ott et al., 2006] and later its mobile extension Rollin' Justin [Borst et al., 2007]. But the aRD framework is also used in DLR's torque-controlled biped humanoid robot [Ott et al., 2010] and the advanced antagonistic humanoid upper body HASy [Jörg et al., 2014]. In all this applications, aRD is mainly used to connect the distributed sensor and actuator resources and to run the distributed control algorithms.

- *Planetary Rover* [LEITE et al., 2012]: For the experimental wheeled planetary rover ExoMars, the communication with all sensors and the distributed control algorithms are realized with aRD.
- *Medical robotics*: aRD is used in the advance robotic surgery setup MiroSurge [TOBERGTE et al., 2009][HAGN et al., 2010][TOBERGTE et al., 2010] to implement the distributed control algorithms for the robots as well as the coupling to the haptic telepresence station. Moreover, it also connects a mobile user interface on an iPhone [14].
- *Brain/EMG based control in assistive robots* [VOGEL et al., 2010][VOGEL et al., 2011][VOGEL et al., 2015]: aRD is used to implement the distributed control algorithms and for coupling a non-realtime interface implemented in Simulink to the realtime control.
- *Biomechanics experiments* [HÖPPNER et al., 2017]: To learn about the biomechanics of the human, test stands with actuators and sensors are developed and controlled by a realtime Simulink model. All the communication is based on aRD.
- *Human robot cooperation* [PARUSEL et al., 2011] [FUCHS et al., 2010]: In a co-worker setup, where a number of torque controlled robotic arms are closely working together with a human to solve manufacturing tasks, aRD is used in the distributed control of the multiple arms as well as in coupling a complex state machine and a non-realtime user interface.

2.6 Summary

In this chapter, we presented the hard realtime capable and highly performant communication layer of our aRDx robotic software framework. aRDx is key for extending the versatility of our humanoid robot Agile Justin, which was built as a research platform for dextrous mobile manipulation, towards complex dynamic mobile manipulation tasks. In general, robotic software frameworks deal with the complexity and high computational demands of deep sensor-perception-planning-action loops using distributed sensor, actor and computing resources by providing a component based system architecture. But it is only due to the high performance of aRDx that allows for such a component based approach also under the tight timing constraints of dynamic tasks.

aRDx's predecessor, the aRD software concept, which we presented in [1], was designed to enable the distributed execution of computationally demanding complex control algorithms for our advanced mechatronic systems with many DOF. aRD also provides a component based software architecture under hard realtime constraints and it provides a tight coupling to Mathworks Simulink/Coder toolchain but it allows only for point-to-point communication and is inefficient for large data packet sizes.

The main insight for the design of the aRDx framework was that two instead of a single communication layer are necessary to cope with the wide range of domains in robotics software from low-level hardware drivers up to artificial intelligence (AI) based decision making. aRDx's higher level communication is, as we sketched in [20], directly based on a modern "programmable programming language" of the Scheme family and allows for the transport of flexible and arbitrarily complex data data structures as used in AI. The lower level communication of aRDx is more static, but it is because of this that allows for its high performance and hard realtime capabilities. We presented this second communication layer in this chapter and in [13] and [3].

There we described the design goals and design considerations for aRDx's communication layer. The most important goals are:

- identical API for intra-process (process domain), inter-process (host domain) and distributed communication,
- hard realtime determinism in the kHz range,
- optimal data transport, i.e., zero-copy for the process and host domain and copy-once to the remote host in a one-to-many communication for the distributed domain,
- detailed control of the quality of service of the communication channels,
- advanced multi-channel synchronization mechanisms.

Our implementation of aRDx reaches all these design goals. aRDx has an elegant hierarchical architecture where everything is based on the basic channel mechanism of the process domain. By mapping this channel to shared memory, host domain communication is realized via the aRDx-daemon for keeping memory protection between the clients. The distributed domain communication is based on these daemons (one on each host) which copy packets once between hosts and provide them locally via zero-copy to the clients. Even the advanced channel synchronization mechanisms are directly built on top of the basic process domain channels.

We performed elaborate stress test benchmarks to compare aRD and aRDx to other most prominent robotic software middlewares, namely ROS, YARP and Orocos. aRD and aRDx perform best for all communication domains and aRDx is the only framework that provides optimally efficient copy-once transport in the distributed domain. When run on the hard realtime OS QNX, aRDx achieves hard realtime determinism for all three communication domains, even when using complex communication schemes including channel synchronization.

In addition, in Sec. 2.1.3 we discussed the recent efforts in other robotic software frameworks to support realtime communication. Especially ROS 2, a redesign of ROS which is based on third party DDS implementations, has hard realtime performance as an important design goal, but even in its latest release the realtime support is only rudimentary.

aRD and aRDx were originally co-developed with DLR's Justin family of advanced torque controlled humanoid robots. And with aRDx, the mobile humanoid Agile Justin [5] reached a new level of versatility in that it can now also perform complex dynamic manipulation tasks. The challenging dynamic task of catching thrown balls with its hands was presented in [6][7] and is also described in detail in Chapter 3.

aRD and aRDx are also used in many other challenging robotic application domains, e.g., surgical robotics [14], planetary rover testbed or human robot co-workers, as we have shown in [1] and Sec. 2.5.3.

Chapter 3

Ball Catching as a Complex Dynamic Task

3.1 Motivation

Catching a flying ball with a hand is a challenging, highly dynamic task, even more so when using only onboard sensing on a moving mobile humanoid robot. But it is also for humans a hard task, e.g., ball catching is used for assessing the motor impairment of children [WAELVELDE et al., 2003].

By realizing ball catching, Agile Justin (see Fig. 3.1) achieves an unprecedented level of versatility in the field of mobile humanoid robotics: it can now perform demanding tasks in dextrous as well as highly dynamic mobile manipulation.

We chose ball catching as a benchmark for a dynamic mobile manipulation task, precisely because it is hard and not forgiving: a tight interplay of fast perception, a good catching strategy, body control and dexterity is needed to achieve the necessary precision in space ($< 2\,\mathrm{cm}$) and time ($< 5\,\mathrm{ms}$), and all has to be done in realtime due to the ball's short flying time ($< 1\mathrm{s}$). Moreover, not only a robotics expert, but even the layman can easily judge the performance of the system by comparing it to human performance.

A key role in realizing ball catching on the advanced humanoid robot plays our aRDx software framework. Ball catching with its complex and deep perception-planning-action-control loop running on distributed computing resources demands for a component based architecture to cope with the system's complexity. But only due to aRDx's high performance and hard realtime determinism, such a component based architecture can be realized despite the tight timing constraints of this dynamic task.

3.2 Related Work

Ball catching has already a quite long history in robotics for testing theories and implementations in as different fields as fast hand-eye coordination [HONG and SLOTINE, 1995], neural networks for human like movement behaviour [NISHIWAKI et al., 1997], off-the-shelf components for realtime vision [FRESE et al., 2001], motion primitives for human like path generation [RILEY and ATKESON, 2002], tele-operation for fast tasks [SMITH and CHRISTENSEN, 2007] or kinematically optimally realtime motion planning [8].

In the pioneering work of [HOVE and SLOTINE, 1991] and [HONG and SLOTINE, 1995], the 4 DOF "WAM" arm, equipped with a gripper to grasp the ball, and an active vision system is used. The catch point is chosen heuristically and a Cartesian path is generated

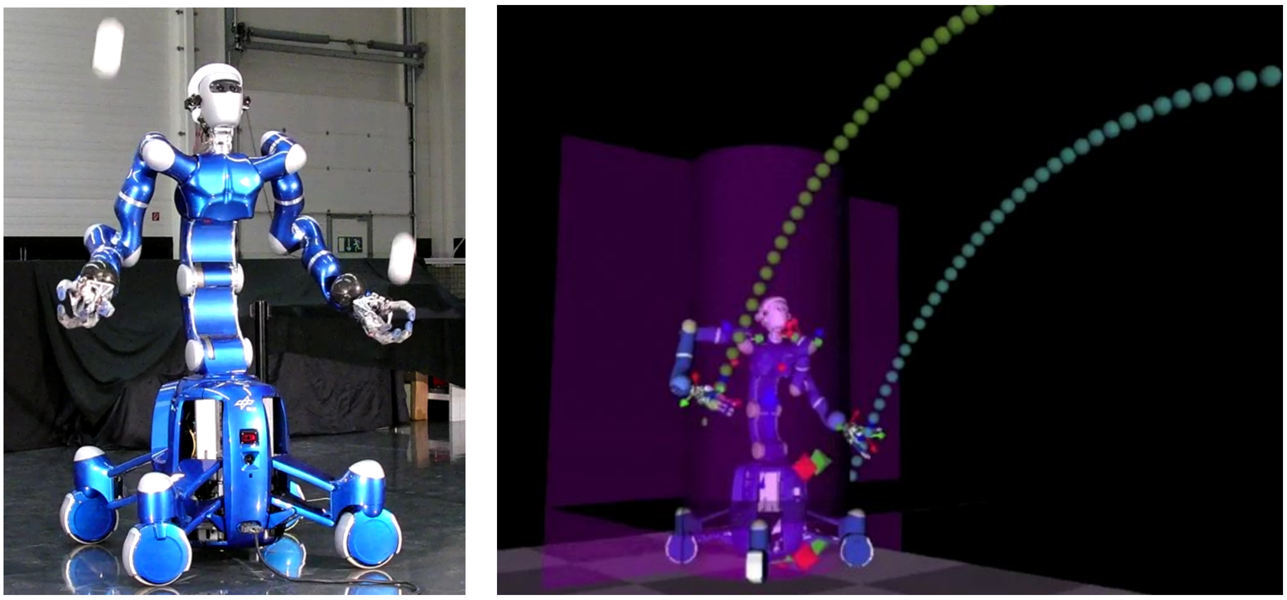


Figure 3.1: *Left:* Agile Justin catching two simultaneously thrown balls. The mobile humanoid operates completely wirelessly using only onboard sensing. All DOF are used to successfully perform the task with a catch rate of over 90%: the arms (each 7 DOF), the torso (3 DOF), and the mobile platform (1 DOF used) are used for the reaching motion, the neck (2 DOF) for keeping the balls in the cameras' field of view, and the hands (12 DOF each) for actively grasping the ball. Velocity limits: arms: $100 - 150^\circ/\mathrm{s}$; torso: $90 - 200^\circ/\mathrm{s}$; neck: $205 - 330^\circ/\mathrm{s}$; mobile platform: 1.4 m/s. *Right:* The ball catching setup. The balls are thrown by a human from a distance of about 4-6 m towards the robot with a speed of typically 7 m/s, resulting in a flight time of about 1 s. The two ball trajectories ($\Delta t = 20\,\mathrm{ms}$) and Justin in the final catch configuration are shown. The (simple) collision model (virtual cylinder and walls), against which the planner tests both TCPs of the final catch configuration to avoid self-collisions, is depicted in pink. The ball has a diameter of 8.5 cm and a weight of 50 g resulting in significant air drag effects (for a 5 m throw $> 20\,\mathrm{cm}$ shorter flying distance compared to a purely ballistic trajectory).

as a third order polynomial and executed by the inverse kinematics running in the control loop.

The seminal work of [FRESE et al., 2001] introduced a system for using common computing hardware for stereo vision processing and catching with a seven DOF lightweight robot arm. A simple catch point selection strategy in combination with inverse kinematics is used for generating the reaching motion.

In [KIM et al., 2014] not only balls but objects with uneven shapes are caught using a stationary vision system, a KUKA LWR 4+ arm and an Allegro hand [BAE et al., 2012]. They use a learning based method both for tracking the object's trajectory and the motion generation. The parameters of a so called dynamical system are learned for arm motions in Cartesian space and a standard inverse kinematics is used to compute the actual joint angles. They demonstrate the successful catching of partially filled water bottles, a tennis racket and a card box.

In [SALEHIAN et al., 2016] the previous method is extended to allow for "soft catching", i.e., instead of having the hand stopped at catch time, it moves with the object for a short period of time. This reduces the impact force and makes the catching more robust as it reduces the relative velocity to the object and leaves more time for the fingers to close on it.

There are only few examples of ball catching with humanoid robots. The system presented in [NISHIWAKI et al., 1997] has a 5 DOF arm on a stationary humanoid upper body, but only the arm is moving. It uses a "cooking basket" at the end effector for catching the ball and an active vision system. The inverse kinematics is solved by a neural network, which should lead to a human like movement behavior.

Riley and Atkeson [RILEY and ATKESON, 2002] presented ball catching experiments using a Sarcos [CHENG et al., 2006] humanoid robot with 30 DOF equipped with a baseball glove. An external stereo vision system is looking for color-coded balls. A Cartesian end effector trajectory to intercept the ball is generated using a programmable pattern generator based on motion primitives. Using an inverse kinematics, the trajectory in joint space is generated.

Disney presented a ball catching setup in [KOBER et al., 2012] using their Animatronic humanoid upper body with 39 DOF and hydraulic actuation. The Animatronic is a robot specialized for human robot interaction in amusement parks and it can not be used for general manipulation tasks where precise motion control and reasonable payloads are needed. For ball catching, they use simple heuristics for catch point selection and an inverse kinematics. The vision system is external and stationary.

In difference to all previously presented work, we show ball catching on a mobile humanoid robot using not only the arms but also the torso and mobile platform for the reaching motion. This makes ball tracking considerably more difficult, as only on-board sensors are used that are now moving.

In addition, we use an unified approach for generating the reaching motion without heuristics and based on nonlinear optimization. This is different to the usual splitting in the three steps: catch point selection, path planning and joint trajectory interpolation. The unified approach allows to work close to the robot's dynamical limits and to define different catch behaviors simply by specifying an objective function.

Only recently, in [KOC et al., 2018] a similar online optimal trajectory generation method was used for another "sports robotics" application, robot table tennis. But there, only a single stationary arm and a non-moving vision system are used.

3.3 Robotic Setup and System Challenges

3.3.1 Setup

Fig. 3.1 and Fig. 3.2 show the overall ball catching setup and Agile Justin's system architecture. The thrown balls are tracked by a head-mounted stereo camera system. Based on this, a (continuously improving) prediction of the balls' trajectories is computed and is sent to the planning module. Then the planner decides where, when, and in which configuration to catch the balls.

3.3.2 Challenges

In what follows, we list the main challenges on all architectural levels in making the mobile humanoid robot Agile Justin catch balls using the arms, torso and mobile platform for the reaching motion.

1. **Low Latency:** Given the flight time of the balls of typically $< 1\,\mathrm{s}$ and the limited dynamical performance of Agile Justin, the robot has to start its reaching motion as soon as possible to cover a reasonably sized *catch space*. Hence, the latency of the tracking, prediction and planning modules should be as low as possible.

2. **High Precision in Space and Time:** To successfully catch a ball, the hand positioning and finger-closing timing has to reach a precision of 2 cm and 5 ms, respectively. This is determined by the hand geometry, ball size and hand closing speed (see Fig. 3.3 for details), so that the ball does not hit the fingers of the open hand and does not bounce out of the hand again before the fingers can close.

3. **Moving Camera System:** To reach the necessary precision in space and time, the ball tracking module has to integrate all measurements of the ball's positions during its flight until immediately before it is caught (because, the closer the ball comes to the robot, the more precise gets the estimation of, especially, the depth). This means, while tracking the ball, the robot is already performing its reaching motion including the torso and mobile platform and, hence, also the head. Therefore, the head motion relative to an inertial frame has to be estimated too.

4. **Not Completely Cancelable Vibrations:** The upper body of Agile Justin is built following lightweight design principles as is necessary for a mobile service robot. But the lightweight structure inevitably introduces elasticities and, hence, vibrations, especially when performing highly dynamical motions. These vibrations can not be completely canceled even when applying elaborate control algorithms [WIMBÖCK et al., 2009]. They are most dominant in the torso joints and make it harder to estimate the movement of the head and limit the precision with which the hand can be positioned, as the path planner can not (easily) anticipate these vibrations.

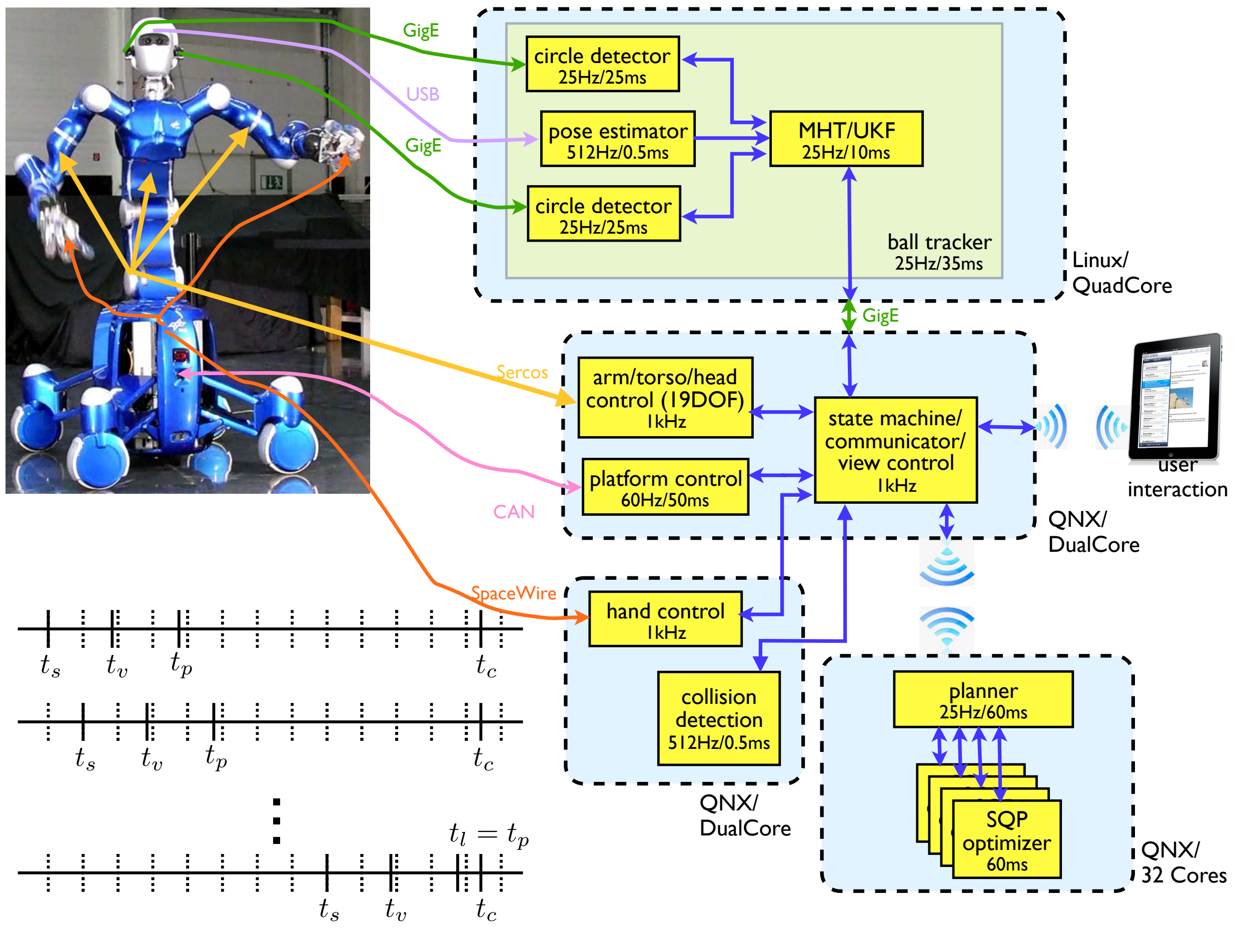


Figure 3.2: Computing, communication and software architecture. Agile Justin's sensors and actuators are coupled by a variety of bus systems to the onboard computing resource, consisting of one Linux (Intel Core i7 QuadCore) and two QNX (realtime OS) computers (Intel Core i7 QuadCore), all connected by GigE network. An external cluster (32 CPU cores, 4x Xeon Dual-QuadCore) running QNX is coupled by WLAN. For all software modules, the rate at which they run, as well as their worst case processing time, is specified. When the cameras (running at 25 Hz) take images at t_s, the data arrives $\Delta t_{v,c} = 40$ ms later at the Linux computer, where the visual ball tracker module is triggered. After processing (worst case processing time $\Delta t_{v,p} = 35$ ms), the resulting ball trajectory prediction is sent at $t_v = t_s + \Delta t_{v,c} + \Delta t_{v,p}$ to the coordinator module. The coordinator collects and distributes all data, controls the whole catching course (e.g., is the system active, catching or idle?), computes the desired head movements (see Sec. 3.5) and communicates with the planning module running on the external cluster. Based on the trajectory prediction, the planner computes the joint paths in $\Delta t_{p,p} = 60$ ms and sends them back to the coordinator, where they arrive at $t_p = t_v + \Delta t_{p,p} + \Delta t_{p,c}$, with $\Delta t_{p,c} = 17$ ms accounting for all delays of the back and forth transfers. The desired paths are then executed and the hand is closed at the planned catch time by means of the controller modules. The diagram in the lower left depicts the timing for two subsequent and the last camera frame ($\Delta t_{\mathrm{fr}} = 40$ ms between dashed ticks).

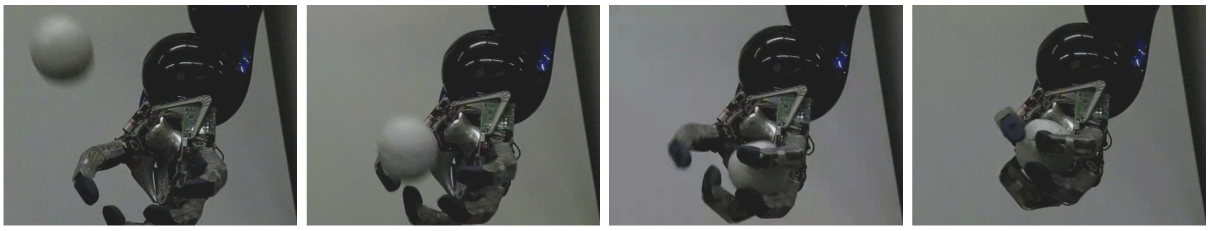


Figure 3.3: Closeup of the hand catching a ball flying with a speed of about 5 m/s (first image immediately before the fingers start to move; $\Delta t = 26.5\,\text{ms}$). Initially, the fingers are pre-shaped to form a little basket with a maximized opening cross section. After the ball has entered, the fingers are closed with maximum speed to finally cage the ball. The *catch frame*, that the planner uses as a virtual tool center point (TCP, one for each arm), lies in the middle of the opening cross section and its z-axis points anti-parallel to the ball's flying direction. This leads to maximum robustness in compensating errors in space and time of the ball trajectory prediction as well as of the arm positioning. Finger parameters: length $L = 14\,\text{cm}$, max. joint velocity $\omega_{\text{max}} = 550°/\text{s}$.

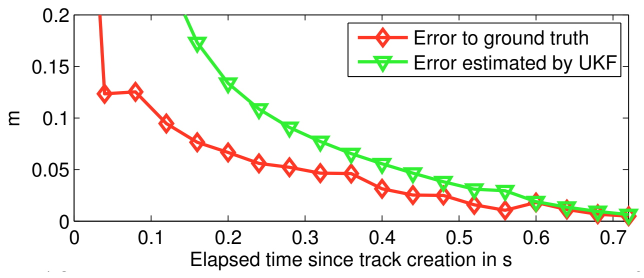
5. **Not Fully Observable Kinematic State:** Besides the observable vibrations, there are non-observable errors between the desired and the actually performed movement: the elasticities in the torso structure lead to "quasi-static" deflections depending on the arm configuration and high accelerations and decelerations of the mobile platform can lead to wheel slippage.

6. **Holistic Calibration of a Multi-Sensorial Humanoid:** To reach the high precision needed (see Challenge 2), the sensors as well as the complex kinematic chain have to be precisely calibrated, e.g., the cameras' intrinsic parameters and mounting frames and the joints' elasticities and offsets. The completely assembled system has to be calibrated as a whole due to inevitable mounting tolerances and the elasticities in the complex kinematic chain.

7. **No Longterm Stability of Calibrated Parameters:** Mainly due to the light weight design of the robot, once calibrated parameters can change over time, e.g., due to slippage and elongation of the torso's tendon or due to the regular maintenance especially at the robot's head.

8. **Limited Computing Resources and Communication Bandwidth:** As illustrated in Fig. 3.2, attention is required what data to compute at which stage and what kind of information needs to be exchanged between components. To operate completely wirelessly, it is required that all computations that demand high bandwidth or low latency have to be performed on the robot's embedded hardware.

In the following sections, we describe the main components we realized for ball catching taking all of these challenges into account.



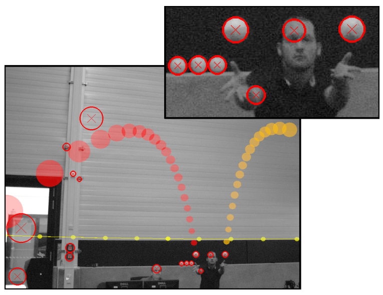


Figure 3.4: Visual ball tracking. *Left*: Prediction error over time compared to the accuracy estimated by the UKF. *Right*: View from one of the cameras mounted at the robot's head while a person is tossing two balls towards the robot. Results are marked in the image: circle detections are depicted as red circles while detected tossed balls are shown as their predicted trajectory through filled circles.

3.4 Visual Ball Tracking and Automatic Calibration

3.4.1 Tracking

The first important stage in catching balls is to track them. In [BIRBACH, 2012] [11] [BIRBACH and FRESE, 2013] ball tracking on Agile Justin is reported that copes with the Challenges 1 through 5.

For tracking balls from camera images, we use a two-staged bottom up approach. First, we detect balls as circles using a scheme similar to Hough-transform but which replaces the hard gradient intensity threshold with a soft illumination invariant indicator of gradient purity. Other than the commonly used gradient intensity, gradient purity is computed via an original normalization scheme based on the local image intensity variance.

In the second step, these circle measurements are fed into a multiple hypothesis tracker (MHT) [COX and HINGORANI, 1996] handling multiple Unscented Kalman filters (UKF) as single target probabilistic models (cf. Fig. 3.4 for a view from the robot including results). Compared to simply fitting a parabola to triangulated 3D points, the UKF takes into account, that measurement uncertainties increase with distance and are larger in depth direction. Fig. 3.4 shows how the prediction accuracy increases over time while the ball is coming closer to the cameras.

The cameras are not static but move when the robot moves and even shake from the reaction forces of moving the arms. Neglecting these vibrations would drastically reduce the precision, roughly by a factor of 3. Our solution to this problem is to view the head-arm system as a self-contained catching device (cf. Sec. 3.5.1). It is somehow moved by the rest of the robot and obtains this motion solely from a head-mounted IMU (inertial measurement unit).

3.4.2 Calibration

Complex mobile manipulation tasks, in particular catching a flying ball, require an accurate interplay of actuation and sensing (Challenge 6). This accuracy can only be achieved by calibrating the relevant components beforehand. In the case of Agile Justin, a pair of

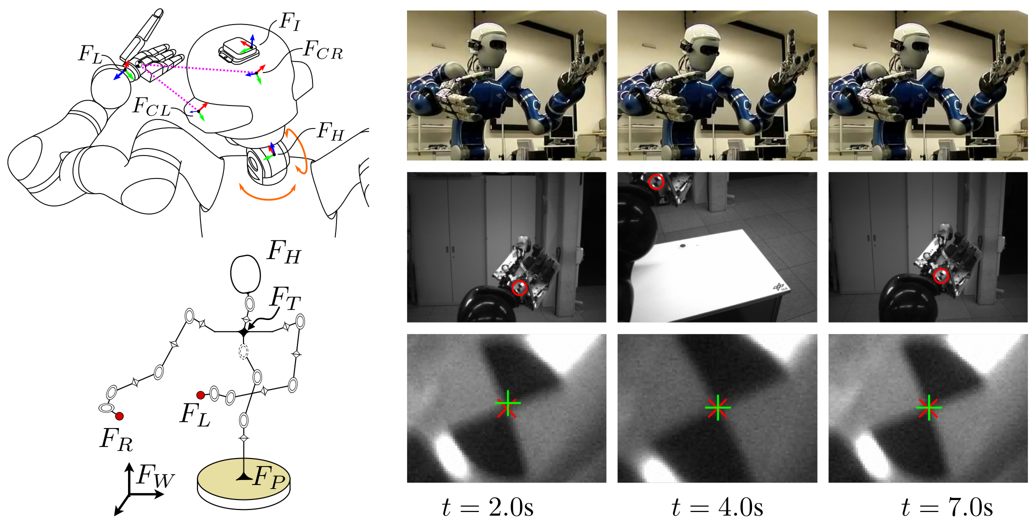


Figure 3.5: *Left lower*: Agile Justin's kinematic with Left and right hand frame F_L / F_R, head frame F_H, torso's chest frame F_T, platform frame F_P, and an arbitrary world frame F_W. *Left upper*: Sketch of the sensor setup. *Right*: The calibration process (Sec. 3.4.2) works as follows: The robot observes (magenta dotted line) a point-feature pasted on the left F_L and right hand F_R (not shown) while moving the head. Additionally, all measurements from the IMU, mounted in the robot's head F_H, and the joint angle and torque sensors are recorded.

stereo cameras, a Microsoft Kinect sensor (not used in ball catching but in other manipulation tasks) and an IMU, all mounted at the robot's head, have to be calibrated with respect to its kinematic chain.

Typical procedures for calibrating are often time-consuming, involve multiple people overseeing a series of subsequent calibration steps and require external tools. We have therefore developed an auto-calibration method for the different sensors of Agile Justin in a single, completely automatic and self-contained procedure. The key idea is, instead of using a few very precise measurements, e.g., by using a calibration plate, to take benefit of having a robot that can autonomously collect at high rate many measurements – although typically with less precision per single measurements.

By automatically detecting a single point feature on each wrist while moving the robot's head, the stereo cameras' and the Kinect's infrared camera's intrinsic and extrinsic as well as the IMU's extrinsic parameters are calibrated while considering the arm joint elasticities and joint angle offsets (see Fig. 3.5). All parameters are obtained by formulating the calibration problem as a single least-squares batch-optimization problem, which is initialized by a rough initial guess, as is, e. g., available when re-calibrating.

The procedure is integrated on Agile Justin and runs completely automatically, from recording the sensor data while moving the head and the arms up to solving the least-squares batch-optimization problem. To speed up the whole procedure, parallel to recording the measurements, the feature detection is already performed (see Fig. 3.6 for the full calibration flow).

In summary, the calibration effort is reduced considerably and allows to obtain an accurate calibration in around 5 minutes by simply "pushing a button".

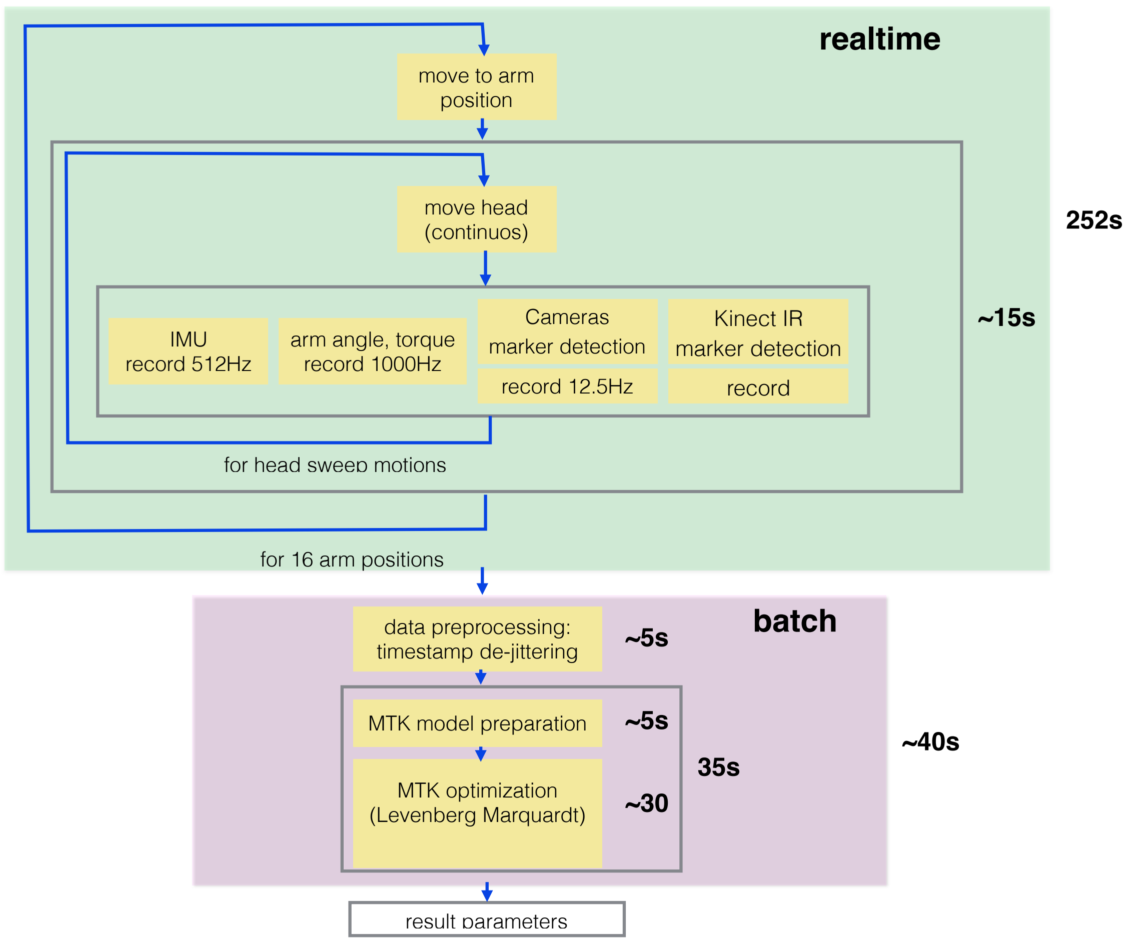


Figure 3.6: Flow chart of the calibration procedure. At the first stage, the robot's arms are moved into the calibration configuration one after another. For each arm configuration, the head is rotated to observe the wrist, while IMU, kinematic data and detected wrist features in the cameras and Kinect IR are recorded in realtime. The second stage batch processes all recorded data by preprocessing (including de-jittering of timestamps), preparing the estimator's model and insert the data and performing the actual optimization to obtain the calibration results.

3.5 Kinematically Optimal Planning

Given the prediction of the ball's trajectory, the path planner has to decide where (catch point x_c), when (catch time t_c) and in which configuration (final joint angles q_c) to optimally catch the ball, while obeying constraints like joint position limits, joint velocity limits or geometrical limits to avoid self-collisions (cf. Fig. 3.1).

We design and implement a kinematically optimal path planner which subsumes all of these three steps in an unified nonlinear optimization problem with constraints. This approach allows to come closer to the robot's dynamic limits, as these limits can be directly included as constraints. But also different catch behaviors can be generated via the objective function, e.g., a "soft" mode, where the joint accelerations are minimal, or a "latest" mode, where the ball is caught at the latest possible point inside the robot's workspace.

Local minima in this nonlinear optimization are overcome by applying a parallel multi-start technique.

Other than the more general, sample-based path planners, like probabilistic roadmaps (PRM) and rapidly exploring random trees (RRT) [LaValle, 2006, Gonzalez-Banos et al., 2006, Karaman et al., 2011], which can cope with geometrically more complex scenes, the optimization-based planner directly computes smooth and optimal joint paths and does so very fast (planning time for a 7 DOF arm $\Delta t_p = 60$ms). The fast processing not only reduces the overall latency (Challenge 1), but also allows to re-compute the paths several times during the ball's flight, which is absolutely necessary as the trajectory predictions of the ball changes significantly during its flight (cf. Fig. 3.4).

3.5.1 Kinematic Subchains

The motion planner uses a rigid body kinematics model of Agile Justin (see Fig. 3.5), ignoring all the disturbing effects as described in Challenge 4 and Challenge 5, such as elasticities, vibrations or wheel slippage. A more precise model, however, would be way too complex for fast planning.

But this problem can be solved by clever use of the kinematic structure of Agile Justin. On one side, the dominant source of errors between the desired and actual motion is the kinematic subchain from the mobile platform to the torso's chest, whereas the two subchains from the hands to the head are very stiff and well modeled by a rigid body kinematics. On the other side, the cameras perpetually measure the balls' positions x_b^H relative to the head frame F_H. So, the precision with which the hand can be positioned relative to the ball at the catch time depends not on the full motion error of the `platform`$\rightarrow$`torso's chest` chain, accumulated between the motion onset and the catch time t_c, but only on the much smaller error, accumulated during the much shorter time span between the time t_l^* of the last ball measurement before the actual catching and t_c! In the worst case, the difference is $t_c - t_l^* = \Delta t_{fr} + \Delta t_{v,c} + \Delta t_{v,p} + \Delta t_{p,c} + \Delta t_{p,p}$ (see Fig. 3.2 for the various time variables). But the timespan can be further reduced, when taking advantage of the direct measurement of the head frame relative to the world via the IMU. Then only the processing time of the planner is relevant and for this t_l time, $t_c - t_l = \Delta t_{fr} + \Delta t_{p,c} + \Delta t_{p,p}$.

This makes it clear that precise planning is only possible for the arms (i.e., `torso's chest`$\rightarrow$`hand` chains), but even so when the imprecise `platform`$\rightarrow$`torso's chest` chain moves. Therefore, for each arm, a motion planner is run for precise hand positioning, whereas the `platform`$\rightarrow$`torso's chest` chain is used to extend the catch space by moving the 3 DOF of the torso and 1 DOF of the platform according to a simple heuristic that brings the hand's start position closer to the predicted ball trajectory. In addition, the 2 DOF of the `torso's chest`$\rightarrow$`head` chain are moved to keep the ball in the cameras' field of view during its flight by continuously pointing the head towards the ball positions according to the first valid prediction from the tracker module. For this, a simple analytic inverse kinematics is applied and a limiter filter enforces the joint position and velocity limits.

The arm planner considers the torso's chest as its *static* base frame. But, because in reality the chest is moving, the ball's trajectory x_b, as predicted from measurements at t_s relative to the head frame, has to be transformed to the torso's chest frame at t_c.

3.5.2 Planning for a Single Arm

When a ball is thrown at $t_0 = 0$, the planner has to solve the following problem: given the start configuration of the arm $q(0) = q_0$ and start joint velocities $\omega(0) = \omega_0$, at which time t_c and in which configuration $q(t_c) = q_c$ with $\omega(t_c) = 0$ should the arm intercept the ball on its trajectory, while obeying constraints? Note that the arm joints do not have to stand still at be beginning, which is especially important to allow for recommanding, as the ball trajectory prediction changes over time.

The following assumptions are made to solve the nonlinear optimization problem in realtime.

Kinematic Planning

Although for catching a ball, the arm has to move fast and, in general, its dynamics plays an important role for generating an optimal movement path, we choose to do a purely kinematic optimization, i.e., the objective function and constraints depend only on q, $\dot{q}$ and $\ddot{q}$ and not on the joint torques τ.

This approximation is not too bad for our LWR-III arm, because for fast motions, the joint velocity limits dominate the robot behavior: e.g., if the robot has to move in 1 s (the ball's typical flight time in our setup) as far as possible, only 0.1 s are needed for accelerating and decelerating respectively, but 80% of the time the robot moves at maximum joint velocity. That means the details of how the robot accelerates/decelerates are not very important for the overall performance, as long as the maximally allowed accelerations are conservatively chosen to never exceed the torque limits of the robot.

Trapezoidal Joint Velocity Ramps

We restrict the robot movements to trapezoidal velocity ramps, because, as argued before, the precise characteristics of the acceleration and deceleration phases are not essential for the overall performance and trapezoidal ramps can be easily handled analytically. Moreover, for the objective functions we present here (see Sec. 3.5.2), the trapezoidal ramps are even optimal.

Using the step function $\theta(t) = \begin{cases} 1 & \text{if } t \geq 0 \\ 0 & \text{otherwise} \end{cases}$, with $\int_0^t dt'\theta(t') = t\theta(t)$ and the definition $\hat{\theta}(t, t_1, t_2) = \theta(t_1 - t) - \theta(t - t_2)$, with $t_1 \leq t_2$, the trapezoidal ramp for a single joint can be written as

$$\begin{aligned} \ddot{q}(t) &= a\hat{\theta}(t, t_1, t_2), \\ \dot{q}(t) &= \omega_0 + at\hat{\theta}(t, t_1, t_2), \\ q(t) &= q_0 + \omega_0 t + \frac{a}{2}t^2\hat{\theta}(t, t_1, t_2). \end{aligned} \tag{3.1}$$

The acceleration phase with $a_{\mathrm{acc}} = a$ ends at t_1, then a phase with $\dot{q} = \omega_{\max}$ lasts until t_2, when the deceleration phase with $a_{\mathrm{dec}} = -a$ starts. There are two cases: first, $t_1 = t_2$, i.e., a triangle ramp, and second, $t_1 < t_2$, i.e., a "full" trapezoidal ramp, where, because of

$\dot{q}(t_1) = \omega_{\max}$, it holds $t_1 = \frac{\omega_{\max} - \omega_0}{a}$. This means that the ramp in both cases is completely determined by only the two parameters a and t_2.

Collision Avoidance

To avoid collisions with the environment and the robot's body, the arm's TCP at the final catch configuration q_c is tested against a simplified but conservative geometrical model as depicted in Fig. 3.1.

Optimization Problem

With these assumptions the nonlinear optimization problem with nonlinear constraints can be written as:

$$(q_c, t_c) = \arg \min_{(q,t) \in \mathcal{S}} H(q,t), \tag{3.2}$$

$$\mathcal{S} = \{(q,t) \in \mathcal{R}^N \times \mathcal{R} : \quad \begin{aligned} & h_i(q,t) = 0, i = 1 \ldots N_h, \\ & g_j(q,t) \geq 0, j = 1 \ldots N_g\}, \end{aligned}$$

with an objective function H, equality constraints h_i and inequality constraints g_j. Note, that the optimization space is only $N+1$ dimensional, because due to the assumptions (especially the trapezoidal ramps assumption), the robot movement is completely determined by specifying the catch configuration q_c and the catch time t_c.

Objective Function

In the following, three objective functions, that lead to significantly different catch behaviors, and the constraints are presented.

- **Soft mode:** This objective forces to make soft movements with accelerations as small as possible for each joint:

  $$H_{\text{soft}}(q,t) = \sqrt{\frac{1}{N} \sum_i^N \left(\frac{a_i}{a_{\max,i}} \right)^2}.$$

  Note, $a_i = a_i(q_i, t)$ is defined by substituting $q_i(t_c) = q_i$ and $\dot{q}_i(t_c) = 0$ in eq. 3.1 and (analytically) solving the resulting quadratic equations.

- **Latest mode:** In this mode, the arm tries to intercept the ball as late as possible on its trajectory, hence

  $$H_{\text{latest}}(q,t) = -t.$$

- **Cool mode**: This is a catch behavior where the robot first moves as fast as possible to a configuration where it can intercept the ball on its trajectory and then "coolly" waits for the ball reaching the hand and finally suddenly closes the fingers to grasp the ball. Unlike for the other catch modes, here one has to distinguish between the time t_c, the robot reaches the catch configuration q_c, and the time t_g, the ball is finally grasped.

$$H_{\text{cool}}(q,t) = \sqrt[4]{\frac{1}{N}\sum_{i}^{N} t_{\min,i}^{4}},$$

with $t_{\min,i} = t_{\min,i}(q_i)$ being the minimal time the joint i needs to reach the given angle q_i when moving as fast as possible, i.e. with $|a_i| = a_{\max,i}$. The minimal time $t_{\min,i}$ is defined by substituting $\dot{q}_i(t_{\min,i}) = 0$ and $q_i(t_{\min,i}) = q_i$ in equation 3.1 and solving the resulting quadratic equation.

Constraints

In what follows, we describe the constraints for the right arm, for the left arm they are analogous.

- **Equality constraints:** To completely define the task – where, when and in which hand orientation to catch the ball – we use equality constraints. At the catch time, the hand has to be at the same position as the ball and the open hand (its z-axis) has to point in the direction of the ball's velocity vector. But the hand's orientation around its z-axis is arbitrary which gives the optimizer more freedom to find an optimal catch motion (see Fig. 3.3).

  Let $F_R(q) = \left(\hat{e}_{\text{x}}(q), \hat{e}_{\text{y}}(q), \hat{e}_{\text{z}}(q), x_R(q)\right)$ be the hand catch frame and $x_B(t)$ and $v_B(t)$ the ball position and velocity at time t. Let further $\varphi_{\text{B}}(q,t)$ and $\vartheta_{\text{B}}(q,t)$ be the angles one has to rotate the vector $v_B(t)$ first around $\hat{e}_{\text{x}}(q)$ and then around $\hat{e}_{\text{y}}(q)$, respectively, so that the resulting vector v_B' becomes anti parallel to $\hat{e}_{\text{z}}(q)$. The equality constraints can then be written as

$$\begin{aligned} x_R(q) &= x_B(t), \\ \varphi_{\text{B}}(q,t) &= 0, \\ \vartheta_{\text{B}}(q,t) &= 0. \end{aligned}$$

- **Inequality constraints:** These constraints force the solution (q_c, t_c) to stay inside the feasible part of the solution space and are defined as follows.
  - *joint angle limits:*
  $$q_{\min,i} \le q_i \le q_{\max,i}, \quad i = 1 \ldots N$$
  - *catch time limits*, with $t_{\max,c} = 1.8\,\text{s}$ for our setup:
  $$0 < t < t_{\max,c}$$
  - *minimal time limits*, which avoid infeasible ramps, i.e. ramps which need a $a > a_{\max}$ or joint velocities $\omega > \omega_{\max}$:
  $$t \ge t_{\min,i}(q_i), \quad i = 1 \ldots N,$$

    with $t_{\min,i}$ as defined in Sec. 3.5.2.

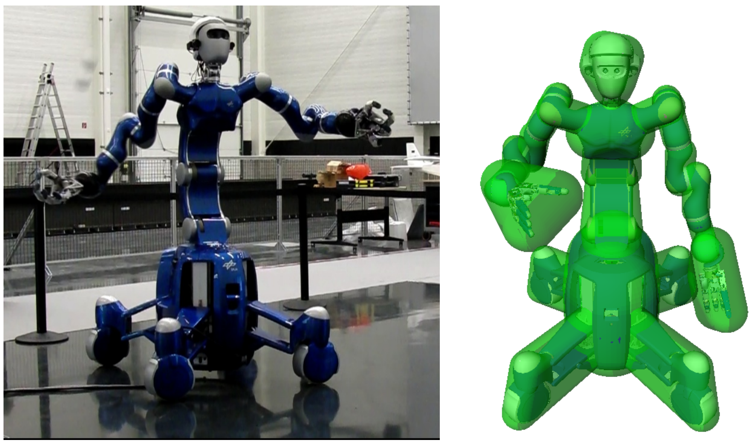


Figure 3.7: Agile Justin and its self collision model made of sphere swept convex hulls (SSCH).

– *collision avoidance*, where $d(x, W)$ is the distance between a point x and one of the $N_W = 8$ geometry objects W as defined in Fig. 3.1:

$$d(x_\mathrm{R}(q), W_k) > 0, \quad k = 1 \dots N_W.$$

In sum, there are $N_h = 5$ equality and $N_g = 2N + 2 + N + 8 = 31$ inequality constraints for the $N = 7$ DOF robot arm.

The solver for the nonlinear constrained optimization problem is based on the highly efficient implementation of the SQP (sequential quadratic programming) method [SPELLUCCI, 1998] [SPELLUCCI, 1999].

3.5.3 Safety Self Collision Detection

When the robot executes the highly dynamic catching motion, any error in, e.g., path planning or control, could lead to self collision of the robot and, hence, to fatal damage. Other than in less dynamic tasks, a manual intervention, e.g., by pressing an emergency button, is not possible. Therefore, we developed a realtime self collision detection algorithm which runs at the full control rate of 1 kHz as a separate component. The self collision module computes, based on the current joint angles and velocities, if the robot could stop using the the maximal deceleration before a self collision would happen. If it predicts a potential self collision, the module initiates a braking motion.

The algorithm is based on computing the swept volumes of all bodies and checking them pairwise for collisions and operates on joint angle intervals. Such, it does not only test a single or N intermediate configurations but assures safety of a whole movement. Key idea of this new swept volume computation is the representation of the volumes as convex hulls extended by a buffer radius, so called sphere swept convex hulls (SSCH). This leads to tight and compact bounding volumes (see Fig. 3.7). The operation set available to model the different joints is strictly conservative and allows for a trade-off between accuracy and computation time. During a configurable timespan the algorithm updates a table of pairwise distances and thus can guarantee hard realtime.

3.5.4 Beyond Ball Catching

Throwing a Ball

Agile Justin can move even fast enough to throw a ball back when combining its arm and the mobile platform DOF in a coordinated whole-body motion (we can not use the torso DOF as their motors are too weak, so we activate these joints' electronic brakes during throwing). We us a similar optimization-based approach as for catching, but now including the additional DOF of the rotation motion around the mobile platform's vertical axis. In addition, a new equality constraint for throwing the ball into the same direction it was thrown towards the robot before replaces the old catch constraint and the objective function is used to maximize the throw distance (Agile Justin can throw up to 7 m).

To successfully throw a ball, also the hand's fingers have to open at the right time, so, throwing demands for a whole-body motion with timing precision in the millisecond range from the wheels to the fingertips. Although the controllers of the hands and the robot's body run on different computers, this timing precision can be achieved due to aRDx's hard realtime determinism even for distributed communication.

The accompanying video[1] of [5] shows an example of Agile Justin playing catch with a human. If there are two Justin humanoids available, they can play catch with each other [Bäuml, 2012][2], i.e., more than 2×20 DOF have to work together in a well-coordinated way.

Optimization-based Path Planning in Self-acquired 3D Models

When planning kinematically or dynamically optimal whole-body motions for tasks like ball catching and throwing, optimization-based planning has proven valuable. But also when, in addition, planning for collision-free paths in geometrically complex environments is required, optimization-based planning has proven successful [Ratliff et al., 2009] [Byravan et al., 2014] and even beats sample-based planners, in particular when it comes to planning for robots with many DOF like our humanoid robot Agile Justin.

For really autonomous operation in previously unknown environments, the robot first has to build a 3D map with its onboard sensors and then plan motions in this self-acquired map. On Agile Justin, we compute in realtime a 3D volume grid model, a so called truncated signed distance function (TSDF) model, of its environment based on its RGB-D Kinect sensor and using a GPU-based mapping algorithm similar to KinectFusion [Newcombe et al., 2011]. From this TSDF voxel model we compute an Euclidian distance transform (EDT) which results in a map where for each voxel the distance to the nearest obstacle is stored (negative distance for voxels outside and positive distance voxels inside the obstacles). For collision free path planning, an obstacle avoidance term is added to the objective function which sums up the positive EDT values of all voxels the robot is sweeping over during its motion. This way, maximizing the objective leads to the robot's trajectory being pushed out of any overlap with the obstacles.

Our main contribution is the idea of computing the obstacle avoidance objective functions and their gradients directly from the KinectFusion model on the GPU without ever

[1]`https://www.youtube.com/watch?v=Fl2N6yZrk1o`
[2]`https://www.youtube.com/watch?v=93WHRSKg3gE`

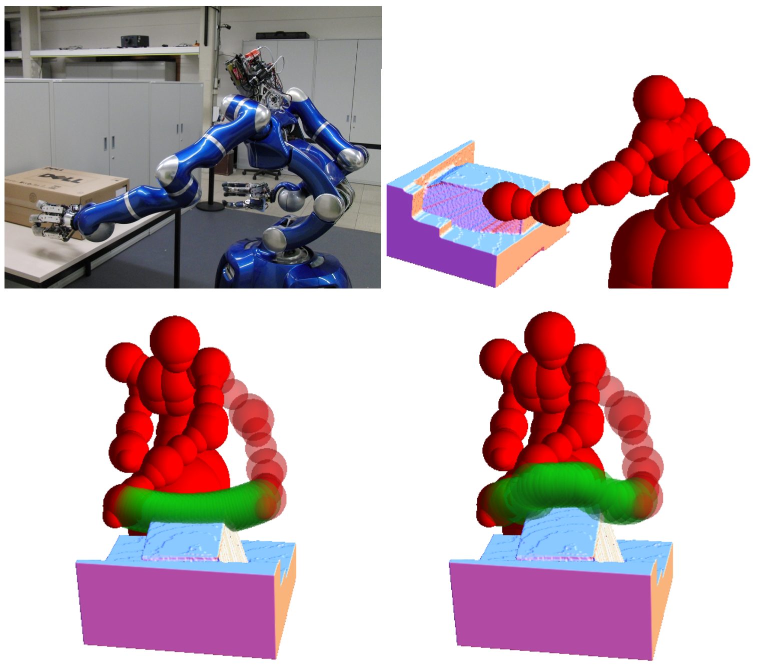


Figure 3.8: Optimization-based whole-body motion planning in self-acquired 3D models. Agile Justin in our evaluation scenario (top left): Its left hand is to be moved from the left of a cardboard box to the right of it. A 3D model of the scene is generated from depth data of a head-mounted Kinect and converted to an EDT representation which is then used for planning (top right). The robot is modeled as a set of spheres (red) positioned according to the joint angles and forward kinematics. The initial path goes through the obstacle (bottom left), the planned trajectory avoids it (bottom right).

transferring any model to the CPU and the integration into a robotic system (see Fig. 3.8). In our current implementation, the summed-up time from taking the first look at the scene until the path avoiding an obstacle is planned, is less than three second.

3.6 aRDx Based System Architecture

The computing, communication and software architecture based on the aRDx framework is the key for the tight interplay of all components and for the low latencies and high timing precision (cf. Challenges 1, 2, 8).

Parallel and Distributed Computing

Even onboard of Agile Justin, parallel and distributed computing resources are necessary: despite their high computational demands, the visual tracking and whole-body control modules have to run onboard because of the high data volume (100 MB/s for both stereo images) and the high control rate (1 kHz), respectively. In contrast, the planner is run on an external cluster, as only the ball trajectory predictions and resulting joint paths have to be transferred back and forth at the cameras' frame rate (25 Hz).

In what follows, we describe how the main modules of the ball catching architecture from Fig. 3.2 are implemented on Agile Justin's computing resources, using the key features of the aRDx framework. The Fig. 2.11 in the previous chapter 2 gives an overview of this aRDx based implementation of the architecture in Fig. 3.2.

Realtime Ball Tracking

The camera and the IMU readers, the pose estimator and the main ball tracking module are all realized as separate aRDx components, running on the same onboard Linux computer. Despite the realtime requirements and the high data bandwidth, such a modular architecture can be realized because of aRDx's zero-copy transport in the host domain and the communication QoS prioritization.

This modularity not only reduces the complexity compared to a monolithic multi-threaded architecture, as we had to use before the availability of aRDx and where, e.g., for the access to the different sensors many third-party libraries with often interfering realtime APIs had to run in the same OS process. But the modularity also allows for the reuse of components, e.g., the same camera readers in combination with separate compression/decompression components are now used for monitoring as well.

Inside the main ball tracker module, aRDx's process domain communication (again, with zero-copy transport) is used to implement the multithreaded tracking pipeline. In particular, the configurable ring buffering for each aRDx channel greatly simplifies tasks such as the data association of asynchronous sensor streams for multi-sensor fusion.

For automatic calibration, aRDx's generic recording tool is used which provides for an arbitrary number of communication channels either in-memory recording or buffered on the fly disk recording. All this is directly based on aRDx's inbuilt ring buffering.

Parallel Path Planning

The ball tracker sends an updated trajectory prediction every 40 ms to the external planning cluster and the resulting joint trajectory is sent back to the robot. Due to aRDx's optimal copy-once distributed transport and quality of service prioritization, the transport of this trajectory data does not interfere with the low priority communication for system monitoring, e.g., the camera images sent to the GUI computers.

The optimization-based motion planner is computationally demanding with about 60 ms to converge in worst case. Therefore, individual planners run in parallel in a master-slave setting on 8 CPU cores for each arm. Given the first ball trajectory prediction, a parallel multi-start search for a globally optimal solution is executed. Then each updated trajectory prediction is assigned to a free planner slave in a round robin scheme to minimize latency. The master-slave scheme is realized with aRDx's hard realtime host domain communication.

Whole-Body Control

The joint trajectory is sent to the computationally demanding whole-body control algorithms which run distributed on two onboard QNX computers. Despite the high control rate of 1 kHz, the distributed execution is feasible due to the hard realtime determinism

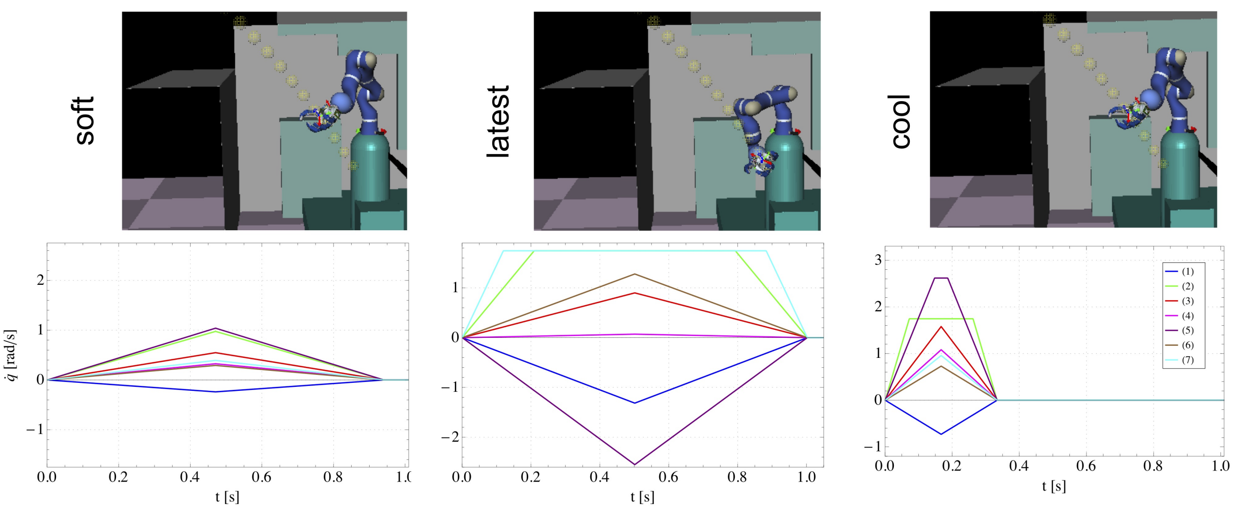


Figure 3.9: Planning results for a simulated ball trajectory in a single arm setup. For each catch behavior, the first row shows the resulting catch configuration q_c and the second row the joints' velocity ramps $q_i(t)$. For "latest" mode, the robot catches the ball later ($t_{c,\text{latest}} = 1.0\,\text{s}$) on the ball trajectory and has therefore to move much faster and further than in "soft" ($t_{c,\text{soft}} = 0.94\,\text{s}$) and "cool" ($t_{c,\text{cool}} = 0.33\,\text{s}$ and $t_g = 0.94\,\text{s}$) mode. The catch configurations for "soft" and "cool" mode are almost the same, but the catch times differ significantly. The smoothest velocity ramps are achieved in "soft" mode, whereas the other two modes reach the acceleration and velocity limits in some joints.

that aRDx's distributed domain communication in combination with QNX's capable network stacks provides.

Also the realtime self collision detection is running onboard and is coupled by aRDx's distributed domain communication to the control components.

3.7 Results

Here, we present the experimental evaluation of the ball catching task. First, for a stationary single arm and then for the full mobile humanoid robot.

3.7.1 Stationary Single Arm

Simulation

To visualize the differences in the three catch behaviors, for each mode the planner was run for the very same simulated ball trajectory and the very same start configuration. The catch behaviors differ significantly and are discussed in Fig. 3.9.

Experiments

The formulation of the subtasks of the motion generation for ball catching, i.e., catch point selection, path planning and joint interpolation, as a unified nonlinear optimization problem without heuristics has two major advantages. One is the definition of the catch

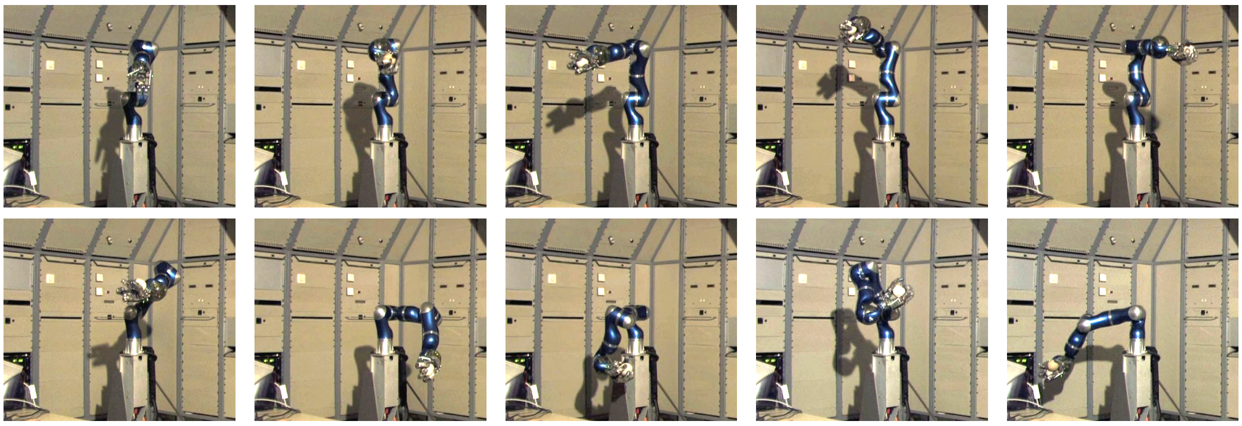


Figure 3.10: Consecutive catch configurations for a continuous catch sequence without moving back to a start configuration before the next ball is thrown ("soft" mode).

behavior by the objective function as shown before. The other advantage is that the start configuration of the arm is arbitrary and the whole workspace can be used for catching. Fig. 3.10 shows an experiment[3] of a sequence of catches where between the catches the arm was not moved back to its default start configuration, but started the new catch from the configuration the catch before ended.

3.7.2 Mobile Humanoid Robot

Fig. 3.11 shows the full humanoid Agile Justin performing ball catching[4]: one experiment with a single ball and one with two balls. In Fig. 3.12 the resulting joint paths for the single ball case are discussed in detail. These two catches and many others can be found in the video [7], both from an outside view as well as from the robot's perspective.

Agile Justin reaches a success rate of over 90 %, given that the balls are thrown inside the robot's catch space.

3.8 Summary

Catching a thrown ball with a hand is a challenging task for humans as well as for robots. This chapter presented the main elements of the holistic system architecture from [6] that enables the mobile humanoid robot Agile Justin to catch up to two thrown balls using only onboard sensing with a success rate of over 90 %. That means, Agile Justin has now reached an unprecedented level of versatility in the field of humanoid robotics: it can perform demanding tasks in dextrous as well as highly dynamic mobile manipulation tasks and, hence, has made a step further towards human versatility.

Key to the realization of the holistic system architecture is the aRDx software framework which allows for a component based approach to cope with the complexity and high

[3]The video of all single arm experiments, which is accompanying [8], can be found at `https://youtu.be/ssR7rIKajeo`

[4]The video from [7], showing the ball catching with Agile Justin, can be found at `https://www.youtube.com/watch?v=R6pPwP3s7s4`

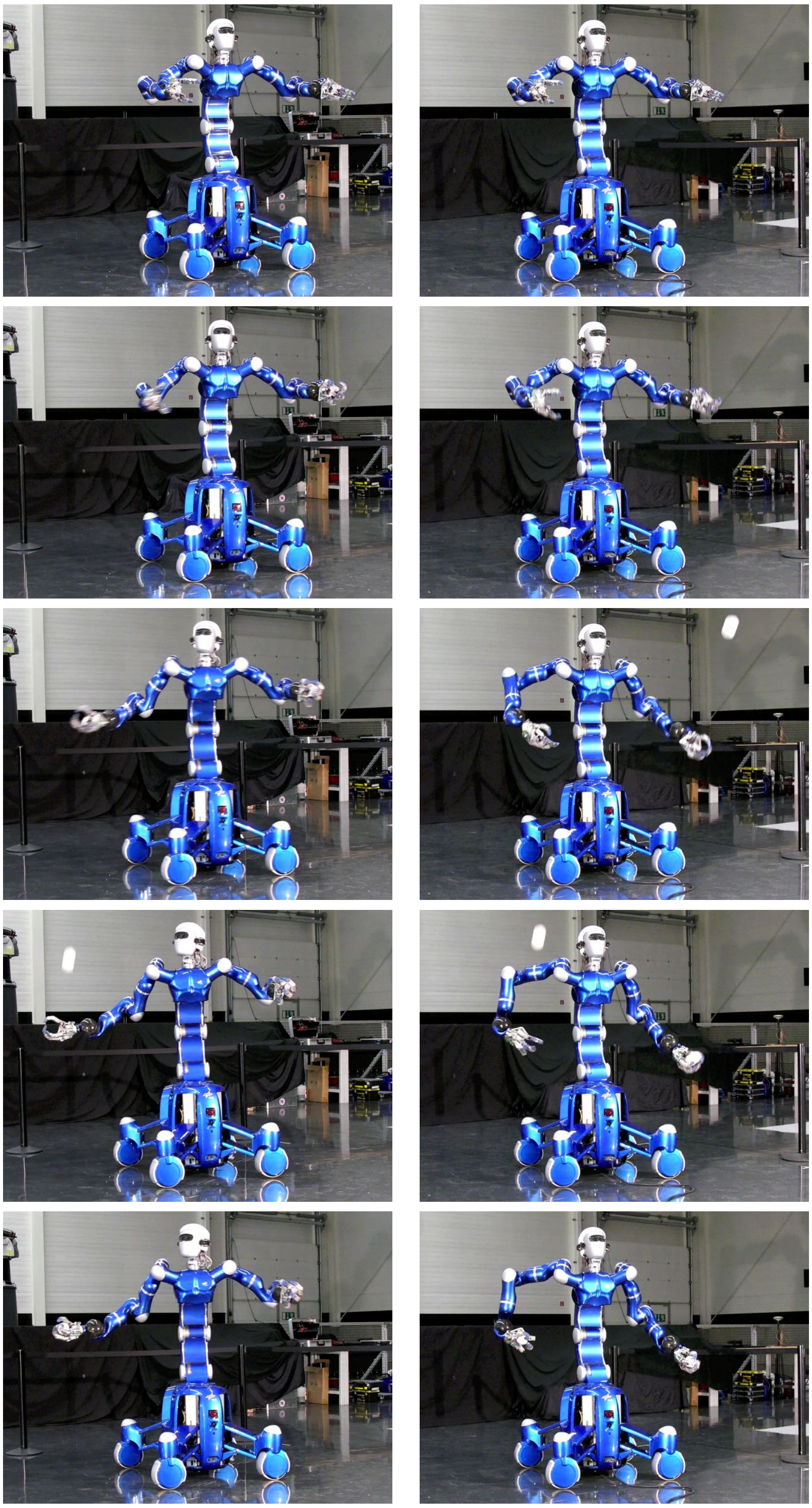


Figure 3.11: Image sequences of Justin catching one (left column) and two (right column) balls (first image immediately before starting to move and $\Delta t = 200$ms). For the single ball case, all DOF are moving, including the mobile platform, the torso and the head, whereas in the case of two balls, only the arms are used.

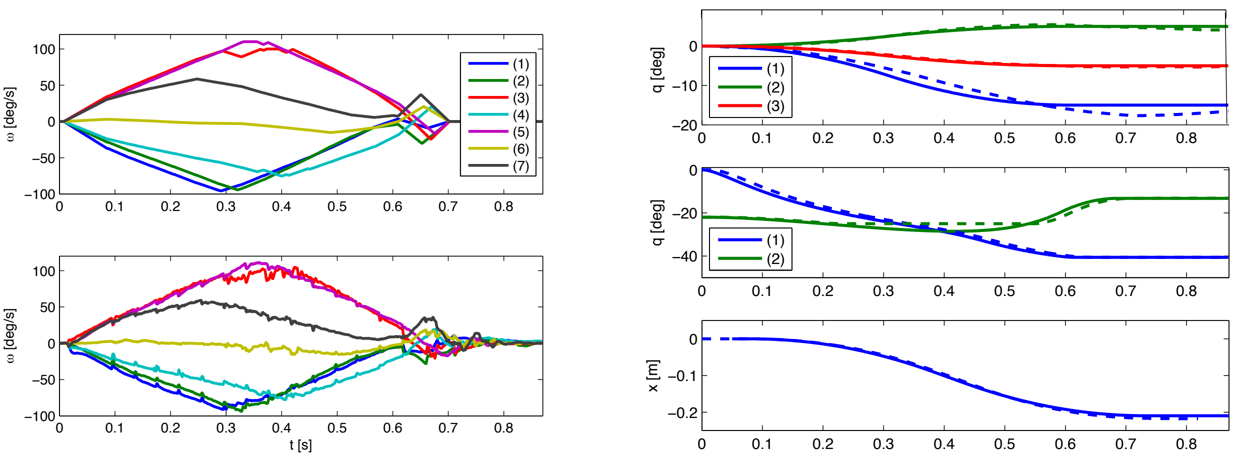


Figure 3.12: Joint velocity ramp while catching a single ball (same as in left column of Fig. 3.11), where all DOF participate in the motion. The robot starts to move at $t = 0$ s, catches the ball at $t_c = 0.702$ s. The last correction movement starts at $t_l = 0.680$ s, which means, that the corresponding planning started at $t_v = 0.603$ s. *Left column*: Velocity profile $\dot{q}_R$ of the right arm. For the desired velocity (upper), the corrections due to the repeated re-planning based on the improving ball trajectory predictions are clearly visible as small kinks. Besides the noise, the measured profile (lower) very precisely resembles the desired one (upper), proving, that the purely kinematic planner and its parametrization generates paths that do not exceed the dynamical capabilities of the robot. *Right column*: Desired and measured (dashed) paths of the torso (upper), head (middle) and mobile platform (lower) joints. The deviations in the head motions are due to the limiter filter and not relevant (as long as the ball is still in the cameras' field of view), as only the measured angles are used in the transformations. The motion errors of the torso and mobile platform, however, directly affect the final precision of the hand position, but only the accumulated error between t_c and t_v (the last time a new head measurement was integrated) counts, resulting in $1.5°$ for the torso's vertical joint (joint 1), which exhibits, as expected, the largest error, and 0.006m for the mobile platform.

computational demands of the deep sensor-perception-planning-action loop despite the realtime constraints and the distributed sensor and computing resources. We described this in [6] and [8] for the original implementation using the aRD framework and in [13], [3] and Sec. 3.6 for the current, more advanced implementation based on aRDx.

In [6] we analyzed the main challenges for realizing ball catching on a mobile humanoid, where "everything is moving" and only onboard sensing and limited computing and communication resources are available.

The visual ball tracking, as described in [11], uses only the head mounted cameras and is based on robust circle detection using a new normalized pattern matching scheme, multi hypothesis tracking and Unscented Kalman filters in combination with IMU (also in the head) based pose estimation to robustly track and predict the trajectories of up to two balls on the moving robot.

As precision is essential for successfully catching a ball, we developed a fast (5 min), automatic ("push a button") and self-contained (no calibration plate) calibration method for Agile Justin's multi-sensorial upper body. In [10] and [2] we presented the calibration method and [12] optimized the set of used arm configurations to reach high precision with a minimal number of configurations to speed up the whole calibration procedure. In [9] we extended our calibration method to also incorporate the second IMU in the mobile base, hence, calibrating a pair of IMUs at the opposite ends of the imperfect kinematic chain of the torso.

For generating the reaching motion of the arms, instead of using separate steps for catch point selection, catch configuration computation and path generation, we presented an *unified approach*, which subsumes all three steps in a single, nonlinear optimization problem with nonlinear constraints [8]. This not only gives rise to a better overall performance (e.g. bigger reachable workspace and faster thrown balls can be caught) as one can get closer to the dynamical limits of the robot, but also allows to optimize for different criteria, which leads to significantly different catch behaviors. In [6] we described how the realtime optimal planning of the arm motion is combined with a heuristics for the motion of the torso and mobile platform joints to increase Agile Justin's catch space by intelligently taking the different kinematic subchains into account.

We use a similar optimization-based approach for planning an optimal whole-body trajectory, including the mobile platform, to make Agile Justin throwing the ball back, as shown in [5].

When developing highly dynamical robotic applications, an independent tool for collision prediction and avoidance is essential to provide safety for the robot. For this, we developed in [15] a realtime swept volume based distance computation for self collision detection for complex serial kinematic chains which was in [16] extended to the mobile platform.

The realized ball catching and throwing system gained respectable attention, resulting in an award winning [5] and an award finalist [7] video contribution, which has more than 460000 views on YouTube[5] and international media coverage. In addition, the two publications describing the collision detection [15] and automatic calibration [10] methods became best paper award finalists at major robotic conferences.

[5]https://www.youtube.com/watch?v=R6pPwP3s7s4

In the future, we want to make ball catching more flexible by allowing for arbitrary and even dynamic environments. For this, we want to adapt our work on GPU based optimization-based whole-body motion planning based on realtime self-acquired 3D environment models using a RGB-D camera [18] [19]. Currently, the complete modeling and motion planning takes less than 3 s, but we are planning to speed it up to 100 ms.

Chapter 4

Deep Learning Based Tactile Material Classification

4.1 Motivation

To allow autonomous robots to robustly and dextrously act in physical contact with their environment, the sense of touch is indispensable. A challenging example is dextrous manipulation with multi-fingered hands, where a feedback signal with high force resolution in combination with a high spatial and temporal resolution is required for the dynamical contact situation. In fact, it is widely accepted that a key prerequisite for closing the large gap in manipulation performance between humans and robots is to come closer to humanlike performance in robotic tactile sensing [DAHIYA et al., 2013] [KAPPASSOV et al., 2015].

A task which can clearly demonstrate the tactile sensing capabilities with respect to force and temporal resolution and, to a lesser extent, to spatial resolution is the identification of an object's material by only gently sweeping over its surface.

4.1.1 Deep Learning for Material Classification

In this chapter, we show that a humanoid robot can exceed human performance in tactile material classification and material differentiation using only the spatio-temporal force signal of a flexible tactile skin mounted on the Agile Justin's soft fingertip (see Fig. 4.1, left). The sensor used is a commercially available and geometrically configurable tactile foil sensor from Tekscan [TEKSCAN] which could be easily mounted (e.g., glued) on the surface of any robotic system. This is an important advantage compared to other tactile sensors, e.g., the bulky sensor like the BioTac [WETTELS et al., 2008] [SYNTOUCH], for which parts of the robot hand's structure have to be replaced, or stiff sensors, e.g., [MITTENDORFER and CHENG, 2011], which can not be mounted on curved or elastic surfaces. Another example of a flexible sensor mounted on a soft surface is iCub's fingertip sensor [SCHMITZ et al., 2010].

To reach superhuman performance, we use a modern deep learning architecture and training methods for processing the complex spatio-temporal tactile signal. We also show that the end-to-end deep learning method outperforms classical learning methods, which are based on manually designed features.

For comparing the tactile performance of humans and the robot, we conduct human performance experiments (see Fig. 4.1, right) on a new representative set of 36 materials

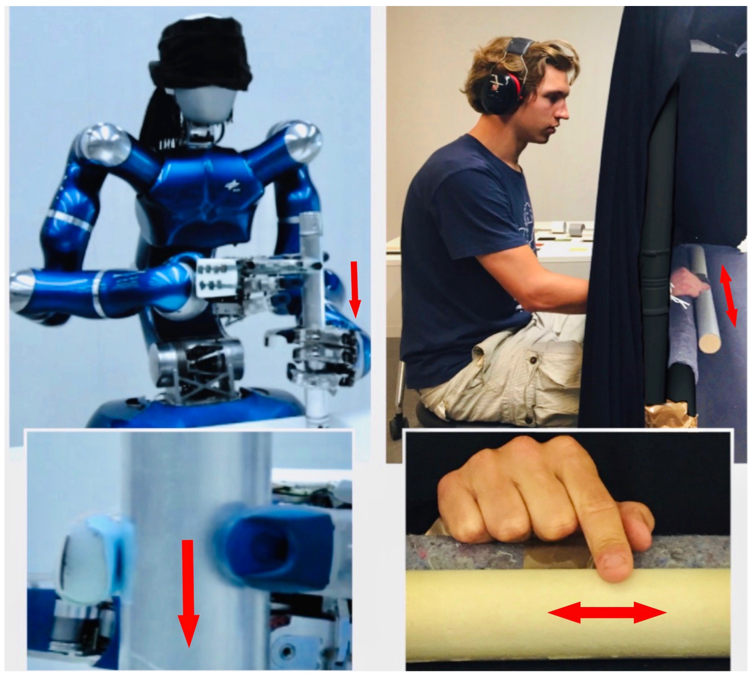


Figure 4.1: Robot vs. human: DLR's Agile Justin (left) and a human subject (right) performing a sweeping motion to identify the material on the surface of a given tube. Only the sense of touch can be used: the visual cue is removed for the human by presenting the tubes behind a curtain and the robot is blindfolded symbolically; the auditory cue is removed by an ear protector for the human (Agile Justin does not have a microphone). Agile Justin is equipped with two DLR Hand-II and to the soft finger tip of the index finger of the right hand a flexible tactile skin with a 4×4 taxel array is attached (see Fig. 4.3 for details). The procedure for exploring a given tube is performed autonomously by the robot: grasp the tube with the left hand to stabilize it; grasp the tube with the thumb and the index finger of the right hand and slide down along the tube with the right hand at a constant velocity of 3 cm/s for 2 s. The force is held roughly at 1 N (precision about 20%) using the hand's joint torque sensors. In the human performance experiment, the subject performs a similar sliding motion with the index finger of its dominant hand, only horizontally on a lying tube.

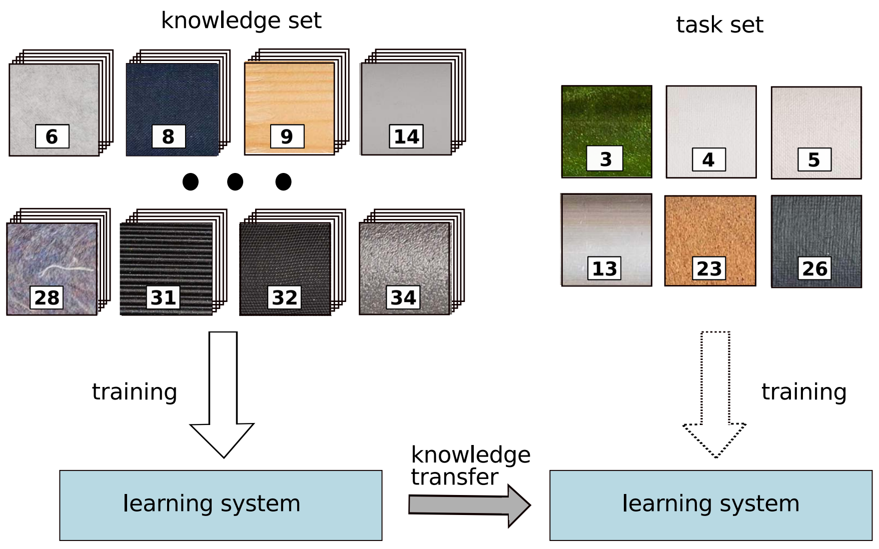


Figure 4.2: n-shot transfer learning. From a large number of classes with many samples per class (the knowledge set) a classifier is trained once. The learned knowledge is then extracted and transferred to a new classifier (often of the same architectural type) to support the learning of a new classification task on previously unseen classes with only a few n samples per class.

typically found in everyday household environments. One task is tactile classification, where the unique identity of the touched material has to be reported. This task measures the more high-level or cognitive performance. The second task is material differentiation, in which pairs of materials are presented and the human subject has to report whether both materials are the same or different. This task measures the raw sensor and low-level processing performance, as no long-term memory is involved.

4.1.2 Deep n-Shot Transfer Learning

A drawback of learning and especially deep learning is that it typically requires a large number of training samples. In robotics, sample acquisition is costly, even more so for active sensing tasks like tactile sensing.

It is therefore desirable to require as little training data as possible and, in the extreme case, to require only a single sample per to be learned class. This is the well known one-shot learning problem [Fei-Fei et al., 2006]. The generalized n-shot learning, where n (small number) samples per class are used, is a topic of intensive research in the machine learning community (see Sec. 4.2).

The main idea in n-shot learning methods is to make use of pre-knowledge to support the learning task at hand. Typically, the pre-knowledge is acquired by training on a large dataset, including many classes and many samples per class, once and then this knowledge is transferred to a new task with previously unseen classes, but only few samples per class. Fig. 4.2 depicts such a transfer learning procedure.

Here, we investigate and show the feasibility of deep n-shot transfer learning for the real world tactile material classification task.

4.2 Related Work

4.2.1 Tactile Material Classification

The seminal work in tactile material classification is [FISHEL and LOEB, 2012], using the multi-modal BioTac [WETTELS et al., 2008] sensor (including static and dynamic pressure, temperature and heat flow). They reported to classify $C = 117$ different materials based on $n = 15$ training samples per material with an accuracy of 95.4% (but needing 5 trial motions on average) using Gaussian classification on hand-designed features. However, the sample data was acquired using a precisely controlled test bench setup and the performance degrades dramatically when transferred to a real robotic setup [XU et al., 2013].

[FISHEL and LOEB, 2012] also conducted a small material differentiation experiment in which five human subjects had to discriminate the materials in each of eight pairs of materials (which were informally selected by the authors from the $C = 117$ materials as the most difficult to discriminate).

[HOELSCHER et al., 2015] used a BioTac sensor on a PA10 robot and performed a comprehensive comparison of diverse feature sets and learning methods for the discrimination of 49 object surfaces. The results show that not elaborate features that depend on the detailed time structure of the signals, but rather simple and "static" features (most prominent temperature and thermal flow) perform best. A linear SVM (support vector machine) in combination with multiple exploration motions reached an accuracy as high as 97%.

[SINAPOV et al., 2011] use a finger nail with an accelerometer as a tactile sensor and performed 5 different scratch motions (different velocities and directions) for discriminating 20 object surfaces. Instead of extracting low dimensional features, they directly used the down-sampled (5 temporal and 25 frequency bins) spectrogram (time varying frequency response) of the 400 Hz sensor signal. Using a SVM classifier, they reach an accuracy of 65.7% for a single scratching motion and 80% using all 5 motions.

We adopt a similar spectrogram based classification method to the tactile skin sensor in Sec. 4.4.1.

[KABOLI et al., 2015] and [CHATHURANGA et al., 2015] address the problem of making tactile material classification based on high-resolution time signals more robust. The former use the 3-axis accelerometers (250 Hz) of their stiff quasi-skin mounted on a Nao robot's body and the latter use a soft 3-axis force measuring tactile sensor (1 kHz) in a test bench setup, where the effect of varying applied force and velocities is systematically studied. Both use correlation based features to dramatically reduce the high dimensional time series (some 100 values) to a low dimensional vector (<10 values) which is then fed into a SVM for classification.

[GAO et al., 2016] present a deep learning convolutional neural network (HapticNet) to classify samples recorded from two BioTac sensors, which were mounted on a robot gripper during five different exploration motions.

For the classification of $C = 14$ materials with a BioTac sensor in a test bench setup, [KERR et al., 2014] reach an accuracy of 79% using surface texture and thermal properties. The authors apply principal component analysis (PCA) to extract important features from the sensor data and a simple neural network for learning them. They also conduct a

Table 4.1: Related work in tactile material classification.
(C = classes; n = samples per class; a = accuracy)

paper	**sensor**	**setup**	**C**	**n**	**a** [%]
[FISHEL and LOEB, 2012]	BioTac	test bench	127	15	95.4
[XU et al., 2013]	BioTac	**ShadowHand**	10	15	99.0
[HOELSCHER et al., 2015]	BioTac	**PA10 arm**	49	50	93.0
[KABOLI et al., 2014]	stiff skin	**Nao**	5	30	100
[KABOLI et al., 2015]	BioTac	**ShadowHand**	20	10	83.5
[SINAPOV et al., 2011]	accelerometer	**robot**	20	50	65.7
[CHATHURANGA et al., 2015]	accelerometer	test bench	8	120	89.0
[GAO et al., 2016]	BioTac	**PR2**	53	10	83.2
[STRESE et al., 2016]	pen	manual	69	188	39.0
[KERR et al., 2014]	BioTac	test bench	14	15	79.1
[EGUILUZ et al., 2016]	BioTac	test bench	34	eff. 300[1]	100
[KERZEL et al., 2017]	OptoForce	test bench	32	100	98.8
[LI and ADELSON, 2013]	GelSight	manual	40	24	99.8
[4]	**flexible skin**	**Agile Justin**	6	80	97.5
[17]	**flexible skin**	**Agile Justin**	36	100	95.0

material classification experiment with 12 human subjects in which the BioTac sensor based accuracy exceeds the human performance.

Multi-channel neural networks are used by [KERZEL et al., 2017] to classify a set of $C = 32$ materials based on the 3D force signals of an OptoForce sensor [OPTOFORCE] and in a test bench setup. They reach an accuracy of 99% but the performance dramatically drops to 68% when artificial Gaussian noise is added to simulate a real world robotic setup.

Table 4.1 summarizes these and more works on tactile material classification. Most works use test bench setups, although those results do not transfer well to the noisy environment of a real robot system. Here we use the humanoid robot Agile Justin, which results in significant variation in the collected tactile samples (see Fig. 4.3).

Our work presented here is the first to exclusively use the signal of a flexible tactile skin for material classification. That means, we show for the first time, that a tactile sensor, which has the aforementioned advantages of providing a high-resolution spatio-temporal signal, as it is required for fine manipulation, and of being easy to mount on any robotic structure, can also be used for material classification.

To our knowledge, there are no other comparison studies of the tactile material recognition performance between humans and *real robotic systems*. But even in case of test bench setups, only the two works of [FISHEL and LOEB, 2012] and [KERR et al., 2014] using the BioTac sensor conduct small experiments for a comparison with human performance.

[1]The 5 min recording time results in effective 300 samples assuming 1 s sweeping time as we use in our experiments.

4.2.2 n-Shot Transfer Learning

To our knowledge, besides ours, the only work on transfer learning in tactile material classification is [KABOLI et al., 2016]. They use the signals of the 2 kHz pressure sensor and the 19 electrodes (50 Hz sample rate) of the BioTac [WETTELS et al., 2008] sensors mounted on the fingertips of a Shadow Hand [SHADOW]. The hand holds the objects while sliding over the surface with two fingers. From the raw high-dimensional signal, first, lower dimensional features are extracted, using manually designed feature descriptors. Their online tactile learning algorithm, which is based on a least squares support vector machine (SVM) classifier, works in this lower dimensional feature space. For 1-shot learning with a knowledge set of 10 objects and a task set of 12 objects, they reach an excellent accuracy of 97 %.

In contrast, our work uses a flexible pressure-sensitive skin instead of the bulky BioTac sensor. We also use a larger dataset with 36 materials which, in addition, is publicly available and allows for comparison to other methods in the future.

The most important difference, however, is that we do not use manually designed feature descriptors, but a deep end-to-end learning scheme with automatic feature extraction. End-to-end learning is important not only because it makes the time-consuming feature design by an expert obsolete, but because the automatic feature extraction gives better performance for the flexible skin sensor (see Sec. 4.4.2).

Recently, there has been much interest in n-shot learning in the context of deep learning, which usually demands for large training set sizes. Early works [HOFFMAN et al., 2014] use feature extraction from a network trained on the large knowledge set and then use classification in this feature space. A more advanced, so called matching method, is the Siamese network [KOCH et al., 2015], in which twin networks are trained especially for getting a good feature space for later matching via a simple distance function, e.g., the Euclidian distance.

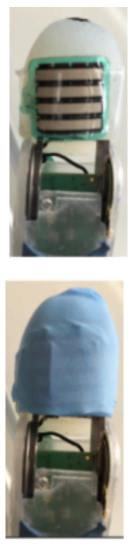
Newer works use quite complex (recurrent) neural networks that require episodic training [VINYALS et al., 2016] [SNELL et al., 2017] [RAVI and LAROCHELLE, 2017]. These transfer learning networks are not easy to train and usually work only for rather simple base network architectures. In contrast, [BAUER et al., 2017] show in their recent work that, using a capable feed forward neural network trained on the knowledge set as a basis, simple matching and logistic regression based transfer methods work surprisingly well and outperform all the more complicated methods by a large margin on the usual benchmark tasks.

Therefore, here we use our high performing deep learning architecture as the basis for the state of the art transfer learning methods from [BAUER et al., 2017].

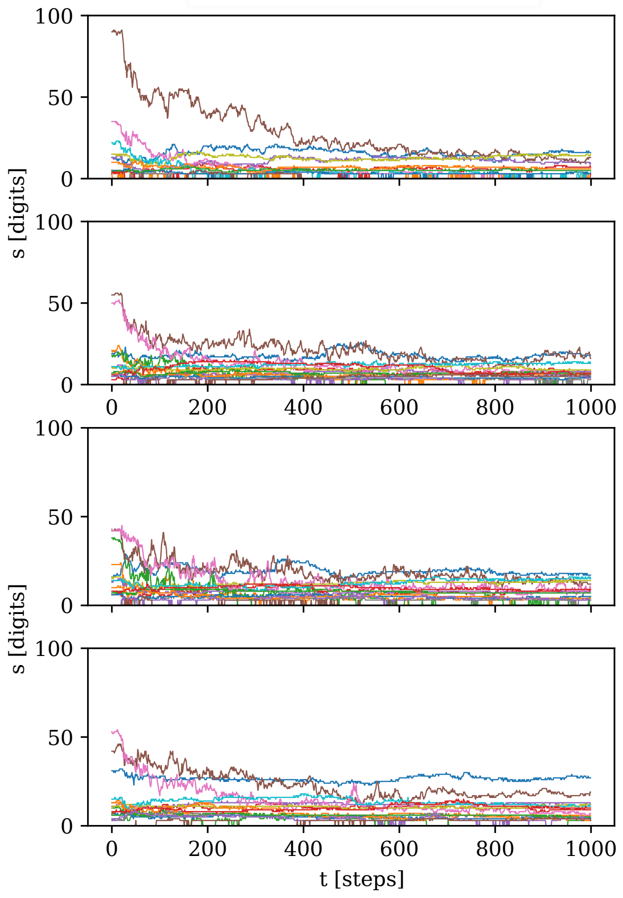
4.3 Experimental Setup

4.3.1 Robot Setup

Our material classification task is motivated by our "Mars Habitat" demonstrator [5], in which the mobile humanoid robot Agile Justin autonomously builds up a scaffold structure from single tubes. The robot should be able to discriminate tubes made of different



a)



b)

Figure 4.3: *a)* The finger tip with the flexible tactile skin taped to it (upper) and the usage of a thin laboratory glove on top of the tactile skin to increase the grip (lower). The tactile skin is a 4×4 taxel array sensor, commercially available from Tekscan sensor 4256E), which provides a spatio-temporal pressure signal at 750 Hz sample rate. *b)* Four plots of the raw spatio-temporal signal of the tactile skin during the middle 1.33 s of the 2 s exploration motion along the tube. The 750 Hz sample rate of the sensor results in a $1000 \times 4 \times 4 = 16000$ dimensional sample, which is used without any pre-processing in our end-to-end deep learning method. To demonstrate how challenging material classification is in this real world robotic setup (in distinction to a test bench setup), the upper two plots depict two samples of the same material class (white metal) which our network correctly recognizes as "same" and the lower two plots show samples for two different materials (cotton fabric reverse and linen fabric smooth) which our network correctly classifies as "different". From looking at the plots, it seems the other way around: the lower two samples look more similar than the upper two.

materials by simply sweeping with its fingers over them, e.g., in case the camera view is obstructed. Fig. 4.1 and the video [BAISHYA and BÄUML, 2016][2] describe the experimental setup for the material classification of tubes with Agile Justin.

We use one sensor patch of a Grip VersaTek® sensor 4256E from Tekscan [TEKSCAN]. The sensor is made from two thin flexible polyester sheets with a printed matrix of piezoresistive ink cells in between. It is only 0.1 mm thin and the patch with 4×4 taxels (size 2.5 mm) is 1.6 cm $\times$ 1.6 cm wide and the pressure range goes from 0.34 kPa up to 345 kPa. An important feature is the high sample rate of 750 Hz for reading out the full sensor.

We attached the sensor patch to the tip of the index finger of the robot's right hand by using a double-sided adhesive tape (see Fig. 4.1 and 4.3). The robot's fingertip is made of soft silicon and has a slightly curved contact surface with the sensor. To increase the grip while sweeping over a surface, we added a layer of latex cut off from a thin laboratory glove. This simple attachment method can be performed easily and quickly and results in a stable position of the sensor relative to the finger's kinematics during long-term use ($> 10^4$ sweeps).

4.3.2 Material Dataset

The $C = 36$ everyday household materials of our sample dataset are depicted in Fig. 4.4. The materials are glued to tubes or the tube is made of the material[3].

The procedure for exploring a given tube is performed autonomously by the robot: grasp the tube with the left hand to stabilize it; grasp the tube with the thumb and the index finger (equipped with the tactile sensor) of the right hand; slide down along the tube with the right hand at a constant velocity of 3 cm/s for 2 s; release the tube. While sliding down, the contact force of the right hand's thumb is controlled to stay constant at about 1 N with a precision of about 20% using the hand's joint torque sensors. For material classification and differentiation we use the 1.33 s in the middle of the sliding motion resulting, given the sample rate of $f = 750$ Hz, in a $1000 \times 4 \times 4 = 16000$ dimensional spatio-temporal signal per sample.

For each material tube, we recorded $n = 100$ samples resulting in $N = 3600$ samples overall. To cover the inhomogeneity of the tubes, each tube was randomly rotated around its vertical axis after each sweep. To cover the drift in the sensor and, especially, in the force control of the hand, a different material was selected each time after 10 sweeps and the recording took place over a period of three days. We made the dataset publicly available at [TULBURE and BÄUML, 2018].

4.3.3 Validation Scheme

Cross-Validation

We use 5-fold stratified cross-validation [KOHAVI, 1995] with 2 runs, i.e., the $n = 100$ samples per class are split in folds of 20 test samples and 80 samples for learning. When

[2]`https://www.youtube.com/watch?v=623yRPx9Pkc`

[3]Only for (9, 10, 13, 14, 19, 20, 34) the tubes are made of the materials.

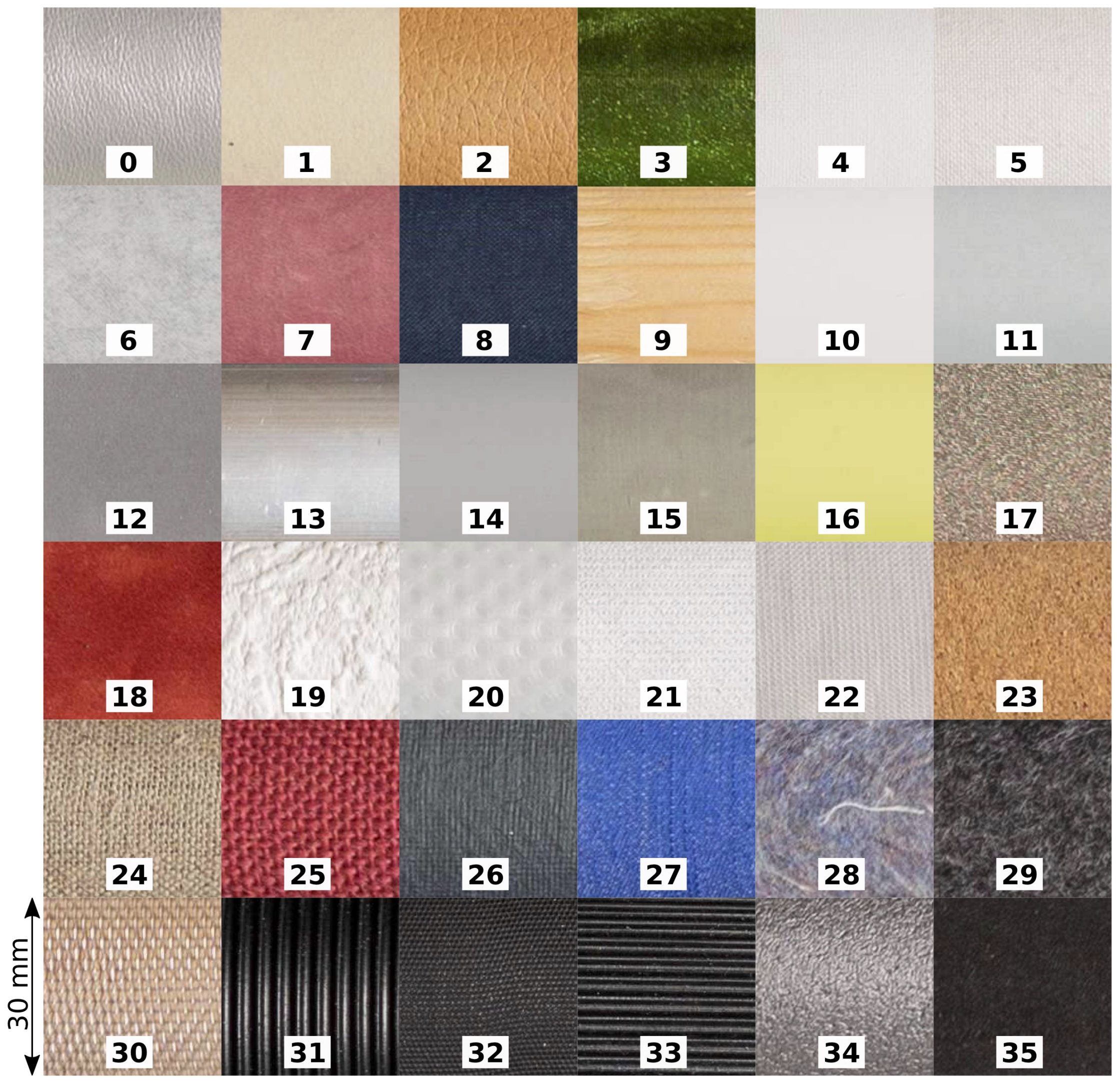


Figure 4.4: The 36 everyday household materials. (0) *synthetic leather rough*, (1) *synthetic leather smooth reverse*, (2) *synthetic leather smooth*, (3) *metallic jersey*, (4) *cotton fabric reverse*, (5) *cotton fabric*, (6) *linen fabric smooth reverse*, (7) *velours paper reverse*, (8) *linen fabric smooth*, (9) *wood*, (10) *white metal*, (11) *reflecting fabric reverse*, (12) *reflecting fabric*, (13) *metal*, (14) *plastic smooth*, (15) *latex*, (16) *silicon*, (17) *jersey*, (18) *velours paper*, (19) *wallpaper*, (20) *plastic rough*, (21) *spun fleece*, (22) *synthetic leather rough reverse*, (23) *cork*, (24) *linen fabric rough*, (25) *gunny*, (26) *carton*, (27) *denim*, (28) *carpet rough*, (29) *carpet smooth*, (30) *metallic grid*, (31) *rubber rough vertical*, (32) *rubber smooth*, (33) *rubber rough horizontal*, (34) *foam*, (35) *neoprene*. Source: Materials 10 and 13 are from Alutruss (www.alutruss.com); 9, 14, 19, 20 from a standard h/w store; remaining from Modulor (www.modulor.de), a shop for designers.

we perform hyperparameter search, the learning dataset with the 80 samples is further split, again using 5-fold cross-validation, into 64 training samples and 16 validation samples. This scheme guarantees that the test samples of a given (outer) fold are never used for training or hyperparameter optimization.

1-sweep & 3-sweep Exploration

The classification accuracy can be increased by sweeping multiple times over the given material. We simulate the multiple sweeps by randomly drawing three samples of the same class from the test dataset and combine the network's per sample prediction with Bayesian fusion (for material classification) or majority voting (for material differentiation).

4.4 Deep Learning for Material Classification and Differentiation

4.4.1 Material Classification

In this section we present our deep learning methods and performance experiments for material classification. For comparison, we first apply classical material classification methods, which have been used for other tactile sensors, to the signal of our flexible tactile skin.

Classical Learning on Manual Features

We start with classical learning methods on two manually designed spectral feature sets for material classification which are computed from the temporal Fourier spectrum of the spatio-temporal skin signal: low dimensional *L-features* and high dimensional *H-features*.

The three L-Features are directly derived from the spectral features reported in [FISHEL and LOEB, 2012]: roughness, which is the integrated spectral power, and two fineness features, which are the spectral centroids in a low and high frequency band.

The 760 dimensional H-features are based on the short-time Fourier transform (STFT) [COHEN, 1995] of the sensor signal, i.e, the frequency response of the sensor over time. This method is similar to [SINAPOV et al., 2011], but we use a higher number of time and frequency bins and extend it to the multi-taxel signal.

On both feature sets we apply two well-known classification algorithms [BISHOP, 2006], namely k-nearest neighbours (kNN) and support vector machine (SVM).

Deep Learning

In difference to classical learning methods with manually designed feature descriptors, deep learning is an end-to-end learning scheme with automatic feature extraction from the raw sensor input. End-to-end learning is important, not only because it makes the time-consuming feature design by an expert obsolete, but because the automatic feature extraction usually gives dramatically better performance as the breakthroughs in

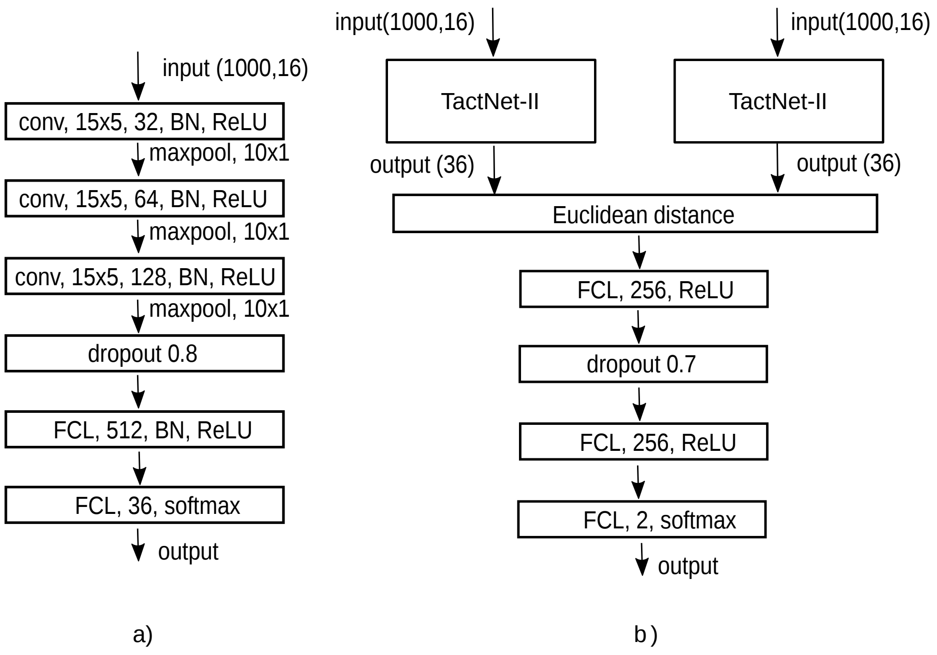


Figure 4.5: *a)* Architecture of the TactNet-II network as used for tactile material classification. *b)* Adapted Siamese network for material differentiation. It is built from two pre-trained TactNet-II networks for mapping the inputs into an abstract feature space where the Euclidean distance is computed.

domains such as image recognition [KRIZHEVSKY et al., 2012] or speech recognition [HINTON et al., 2012] have proven.

Here we apply deep learning to perform material classification directly on the raw 16000 dimensional spatio-temporal signal of the flexible tactile skin.

Base network

In our convolutional neural network (CNN) architecture, the 3D spatio-temporal signal is first converted into a 2D input signal by flattening the spatial dimensions into one dimension, as tests have shown that using the full 3D signal results is no better performance but is computationally less efficient in the here used deep learning software framework TensorFlow [ABADI et al., 2015]. This means that, as expected[4], the temporal correlations are important but that there are no significant spatial correlations in the signal.

The spatially flattened 2D signal is fed into a stack of convolutional and max-pooling layers which implicitly perform the feature extraction. This transforemd signal is used as the input for a fully connected layer followed by a softmax layer for the classification. In each convolutional layer, batch normalization [IOFFE and SZEGEDY, 2015] is performed before the application of the activation function. This allows for a more robust training irrespective of the variation in the samples' statistics.

[4]The spatial resolution of the sensor is way too low to see the fine structure of the materials, whereas the spatial fine structure is, via the sweeping motion, transformed into a temporal structure which the individual taxels can resolve due to the high sample rate.

Fig. 4.5 depicts this base network for which in an architectural search, the number of convolutional layers, the kernel size, and the size of the fully connected layers were optimized using the cross-validation scheme from Sec. 4.3.3.

The network is trained by minimizing a standard cross entropy loss function with a L2 regularization using an Adam optimizer [KINGMA and BA, 2015] with learning rate scheduling.

In what follows, we present two advanced training methods which we adapted for our tactile network architecture.

Adversarial training

Adversarial training [GOODFELLOW et al., 2014] was originally developed to make a classifier more robust against adversarial attacks. But it can also be seen as a smart technique for efficient data augmentation with random noise. In random noise data augmentation, for each training sample x, a number of perturbed samples $\tilde{x} = x + \epsilon\eta$ are added to the training set where the same class label t as for the original sample x is assumed. $\eta \in [0, 1]$ is a uniformly distributed random variable and ϵ is the noise scaling factor (e.g., in the order of the sensor noise). This data augmentation makes the classifier robust against typical real world perturbations and, hence, is a kind of regularization.

The trick in adversarial training is, instead of augmenting with many random samples (which is very inefficient in high dimensional input spaces), to only add the worst case perturbation $\tilde{r} \leq |\epsilon|$, i.e., the perturbation which changes the per sample loss function $E(x, t, \theta)$ for training the classifier's parameters θ the most. The adapted loss function for adversarial training reads then

$$\begin{aligned}\tilde{r}(x, t, \theta) &= \epsilon \, \mathrm{sign}(\nabla_x E(x, t, \theta)) \\ \tilde{E}(x, t, \theta) &= \alpha E(x, t, \theta) + (1 - \alpha) E(x + \tilde{r}(x, t, \theta), t, \theta),\end{aligned}$$

where α is a weighting factor for the contributions of the original and adversarial sample and set to $\alpha = 0.5$ for all our experiments.

Monte Carlo Dropout Model

The Monte Carlo dropout model (MC dropout) [GAL and GHAHRAMANI, 2016] is a recent method for efficient Bayesian learning in deep neural networks. MC dropout reinterprets in a variational inference scheme the usual standard dropout as drawing samples of the network weights W from an approximate posterior distribution $p(W|X, T) \approx q_\phi(W)$ given the training samples and labels (X, T). The resulting loss function for optimizing the parameters ϕ of the approximator $q_\phi(W)$ is exactly the same as the loss function for standard learning with dropout. But during prediction, the dropout is kept switched on and for a given input x, multiple runs M through the network are performed (each with a new $W \sim q_\phi(W)$ via dropout). This results in a Monte Carlo approximation of the full Bayesian predictive distributions

$$\begin{aligned} p(t|x, X, T) &= \int p(t|x, W) p(\theta, X, T) \mathrm{d}W \\ &\approx \sum_{m}^{M} p(t|x, W^{(m)}), \text{ with } W^{(m)} \sim q_{\phi^*}(W^{(m)}). \end{aligned}$$

The exact Bayesian predictive distribution would represent the correct prediction uncertainty, i.e., combined model uncertainty and noise, and would not suffer from overfitting. But the MC dropout approximation usually gives also better prediction than the point estimate of standard non-Bayesian deep learning. An important parameter is the dropout rate which we optimize in the hyperparameter search. The number of runs is set to $M = 100$ for all our experiments.

4.4.2 Classification Results

Method Comparison

Table 4.2 reports the accuracies for the network architectures and training methods on our dataset with 36 materials, using the evaluation method as described in Sec. 4.3.3. When using the advanced adversarial training and MC dropout training methods, the performance is best and an accuracy of up to 86.3 % is achieved. This clearly proofs that tactile material classification with a flexible tactile skin is feasible. We name the network architecture using the advanced training methods TactNet-II. In the 3-sweep case, we use the Bayesian fusion scheme from Sec. 4.3.3 and the accuracy gets as high as 95.0 %.

Table 4.2: Material Classification Performance

sweeps	**network type**	$\bar{\mathbf{a}}$ [%]	σ [%]
1	base	85.5	1.5
	adversarial	86.1	1.1
	adversarial + MC dropout (**TactNet-II**)	**86.3**	1.2
3	base	94.3	1.5
	adversarial	94.5	1.4
	adversarial + MC dropout (**TactNet-II**)	**95.0**	0.9

Fig. 4.6 shows that TactNet-II clearly outperforms the classical classification methods by a large margin. This proves that end-to-end deep learning is superior to learning on manually designed features and that deep learning is the key for making a flexible pressure-sensitive skin, despite its complex and noisy spatio-temporal signal, usable for highly sensitive tactile perception.

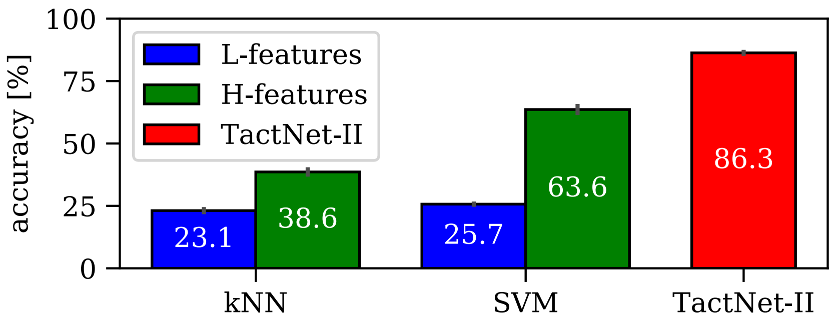


Figure 4.6: Comparison of deep learning TactNet-II to classical classification methods with manually designed features.

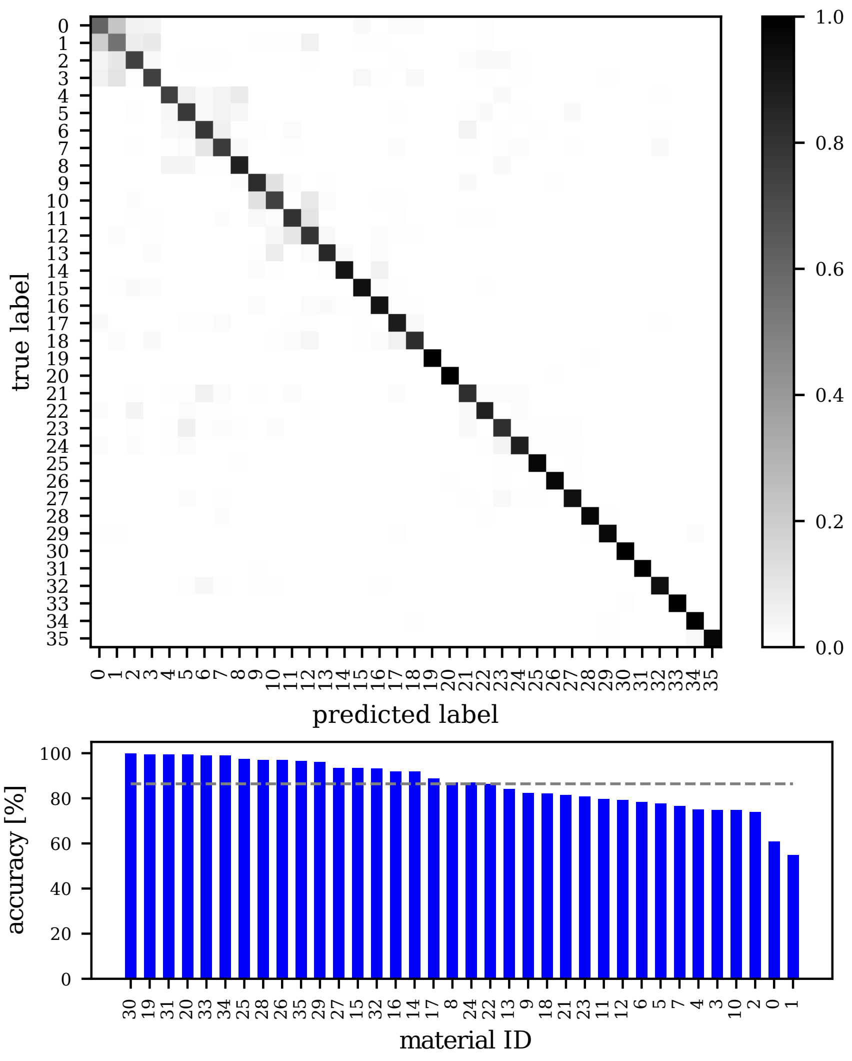


Figure 4.7: Confusion matrix (upper) and its diagonal values (lower), hence, the per class accuracy. For the latter, the order is from high to low accuracies. All results are for TactNet-II and the 1-sweep case. The dashed line marks the average accuracy of 86.3% over all materials.

Confusion Matrix and Grouping

The confusion matrix C in Fig. 4.7 and its diagonal values show that some materials are easier (e.g., the rubbers) and others harder (e.g., the leathers) to classify than average.

For further analysis, we use spectral clustering on the confusion matrix C by setting the affinity matrix to $A = \frac{1}{2}((C-1)+(C-1))^{\mathrm{T}}$ to get 6 groups of materials which are "confused the most" with each other. The materials are then ordered such, that for each group the materials in a group have consecutive material IDs. Actually, this order is used in all our figures, including the Fig. 4.4 of the materials and the Fig. 4.7 of the confusion matrix. In the latter, one can clearly see that the hard to identify materials (especially the leathers) get only confused with one another.

No Overfitting

The standard deviations σ we report are computed via cross-validation and represent the combined uncertainty due to potential overfitting and due to the finite number of test samples per fold. In our results, the σ is no larger than expected due to the finite number of test samples, showing that there is no significant overfitting.

4.4.3 Material Differentiation

In material differentiation, we want to learn a function $f(x_a, x_b)$ which takes two input samples x_a and x_b and reports back 1 if both samples are from the same class, i.e., $t_a = t_b$, or 0 if they are from different classes, i.e., $t_a \neq t_b$.

For this, we adapt a Siamese network model [Koch et al., 2015] which was originally developed in the context of one-shot learning. The idea is that both input samples are first independently mapped into an abstract feature space and then a distance in this feature space is computed between the input samples. Finally, the distance is mapped with an additional network to the decision probability for "same" or "different".

Fig. 4.5 shows our Siamese network model for material differentiation. The two sister TactNet-II networks are identical and have been pre-trained on the classification task and their weights are fixed. Only the weights of the fully connected layers after the Euclidian distance layer are trained specifically for the differentiation task.

4.4.4 Differentiation Results

For the performance evaluation of our material differentiation network, we first perform the usual cross-validation split into test and training samples. Then, for each of the two datasets, we randomly generate sets of material pair samples with an equal number of "same" and "different" pair samples ($3.6 \cdot 10^4$ samples for training and $2.7 \cdot 10^3$ for testing).

Table 4.3 reports the performance results. For the 1-sweep case, the accuracy in material differentiation is about 5 % larger than for material classification, showing that, as intuitively expected, differentiation is the simpler task.

Again, we are not suffering from severe overfitting as σ has roughly the size as expected due to the finite number of test samples.

Table 4.3: Material differentiation performance

sweeps	**$\bar{a}$** [%]	σ [%]
1	91.8	0.9
3	95.4	0.9

4.5 Human Performance Experiments

To have a baseline for our robotic tactile sensor and processing performance, we performed human performance experiments for material classification and material differentiation. Fifteen human subjects, eight males and seven females with age between 21 and 49 years, participated in the experiments. All participants were tested to have normal touch sensitivity using the "Touch-Test Sensory Evaluation" from North Coast Medical[5].

Like in the robotic case, the human subjects should only use tactile information by sweeping a finger over the tubes, but no other sensorial cue. For this, we used the experimental setup in Fig. 4.1: the tubes were presented to the human subject behind a curtain to remove the visual cue. In addition, the subjects had to wear ear protectors to remove the auditory cue, as it turned out that humans can hear for some materials the material identity while sweeping their finger over it.

For each subject, the experiments were conducted on two days, on the first day the material classification and on the second day the material differentiation experiment. All experiments were conducted by the same investigator.

4.5.1 Material Classification

The experiments consisted of a training and a testing phase. In the training phase, the subjects had 10 min to get familiar with the materials by touching and sweeping over material samples attached to small plates. During this phase, the materials had to be grouped in five to seven groups according to their subjective tactile similarity. This should help later when performing the tactile classification task. All subjects reported that the 10 min for this phase was more time than they needed. This may have been due to the fact that our material set consists of everyday household materials with which the test persons were already familiar. During the training phase, the visual cue could not be excluded as the subjects had to see where to find the material samples and had to reorder them.

The testing phase consisted of three directly successive sub-phases for each material:

1. Sweep once over the presented tube and decide for the material class by looking at the previously grouped material sample plates and telling the number written on them.

2. Sweep three times over the tube in both directions and decide again by looking at the sample plates for the material class.

[5] `www.ncmedical.com`

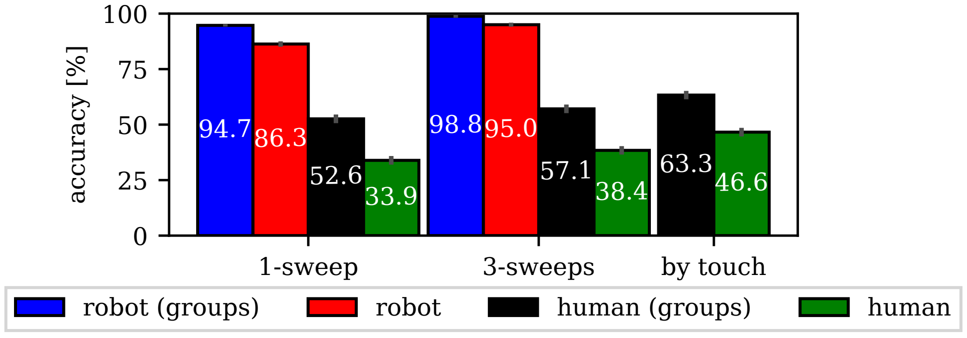


Figure 4.8: Human vs. robot material classification accuracy. Results are reported for the 1-sweep and 3-sweep cases for the robot and the human. For the human, the additional "by touch" case is reported, in which the human was allowed, after having performed the three sweeps over the tube, to also touch the sample materials before deciding for the material class.

3. Sweep ones more over the tube and decide by sweeping over the material samples for comparison by touch.

During testing, each material was presented to the subject once, but the subjects were not told about this, and the order of the materials was random. The overall time of an experiment was at maximum 45 min.

4.5.2 Classification Results

Fig. 4.8 reports the accuracies of the human experiment averaged over all subjects in comparison with the robot performance. The human performance is dramatically worse. Even the 1-sweep robot accuracy is 40% higher than the "3-sweep and compare by touch" accuracy of the human. To make the task even simpler, we also report the accuracy for identifying at least the correct material group which the subjects formed individually in the training phase. But even this accuracy is still about 30% worse than the one of the robot for the way harder 1-sweep class identity task. For completeness, Fig. 4.8 also reports the robot's performance in identifying the correct group using the groups from Sec. 4.4.2.

Fig. 4.9 shows the confusion matrix and the accuracy for each material class. The comparison with the results of the robot in Fig. 4.7 shows that for every single class the robot reaches at least human accuracy.

This surprisingly bad human performance is compatible with the statements of almost all human subjects: after the training phase (in which they could see the material samples while touching them), they expected the classification task to be way easier than they judged it after they had actually performed the experiment (but were not told about their performance). One explanation for this initial overrating of their tactile capabilities could be that humans almost always use additional visual cues to prime their tactile expectation.

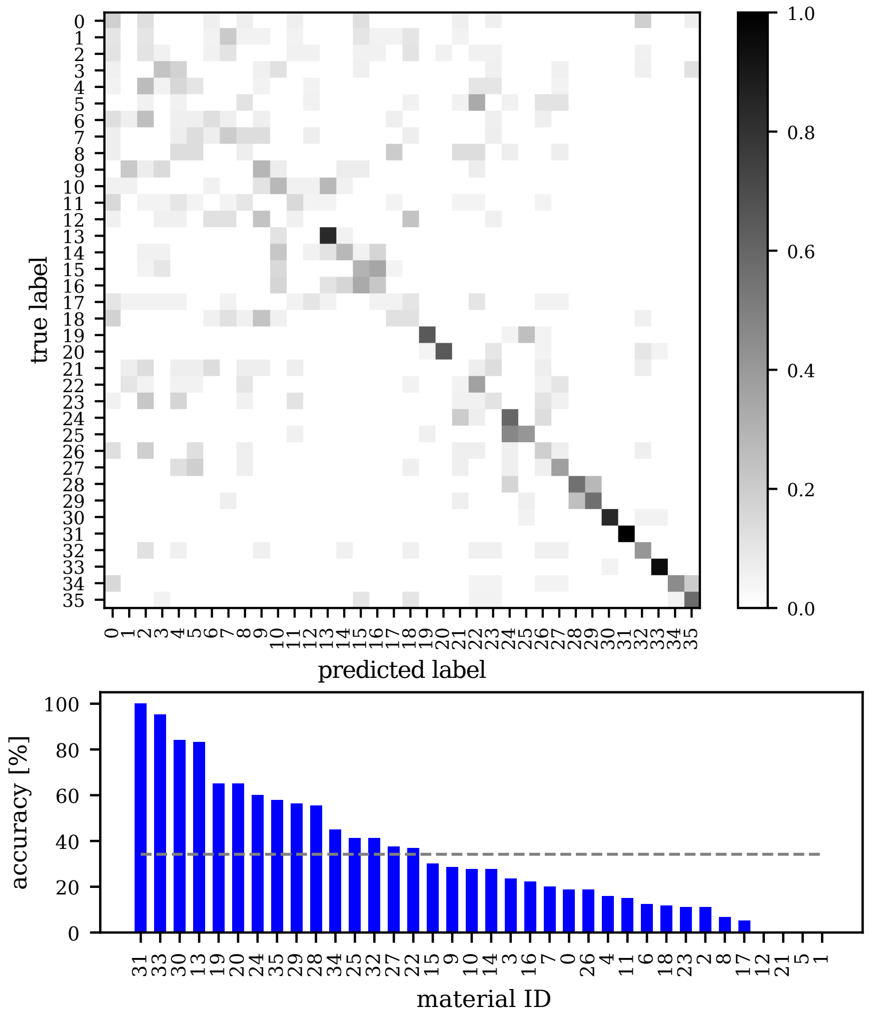


Figure 4.9: Confusion matrix (upper) for the human performance experiment and its diagonal values (lower) ordered from high to low accuracies. The dashed line marks the average accuracy over all materials.

4.5.3 Material Differentiation

Due to the high number of $\binom{36}{2} = 630$ possible material pairings, an exhaustive evaluation of the material differentiation performance in a study with human subjects is impossible. Therefore, to get at least a lower bound for the material differentiation performance, we selected the eight hardest to differentiate material pairs. To have a fair set of the hardest pairs, we selected the four hardest pairs for the human and the four hardest pairs for the robot, using the following two criteria based on the classification confusion matrices:

- The hardest materials are the ones which have the smallest on-diagonal values.
- The hardest materials are the ones which have the highest off-diagonal values.

For each criterion, we selected two materials from the robot and human confusion matrix. Because one material pair was the same in the robot and human case, only seven actual pairs were used in these experiments: (0, 1), (12, 7), (16, 10), (10, 14), (9, 10), (11, 12), (4, 8). These seven material pairs are made up from 11 different materials. In the experiment, we presented the human subjects an equal number of those "different pairs" and of "same pairs" made up from the same 11 materials.

In this experiment, there was no training phase, but the subjects were allowed to make them familiar with all material samples again. The subjects were not told that only pairs from a subset of the materials will be presented in order not to bias their decision.

In the testing phase, a pair of tubes was presented to the subjects behind the curtain. Each presentation had two sub-phases:

1. Sweep once over each tube of the given pair and decide and say if the materials are the same or different.
2. Perform two additional sweeps in both directions over the first and then two sweeps over the second tube. Finally, it was allowed to sweep once more over the first tube before the decision had to be made.

Each of the "different pairs" were presented twice and an equal number of the "same pairs". The presentation order of the these pairs was random. The overall time of an experiment was at maximum 45 min.

4.5.4 Differentiation Results

Fig. 4.10 reports the human accuracy for the differentiation task in comparison to the robot performance for the selected hardest material pairs. To show that this accuracy is a lower bound for the accuracy computed over all pairs, also the robot accuracy for all pairs is depicted.

The robot clearly outperforms the human in material differentiation, although by a smaller margin than for material classification. An interpretation of this finding could be that the human raw sensorial and low-level processing tactile performance is good, but that human tactile memory is not well trained. It would be interesting to repeat the experiments with a blind subject which is more dependent on its tactile performance.

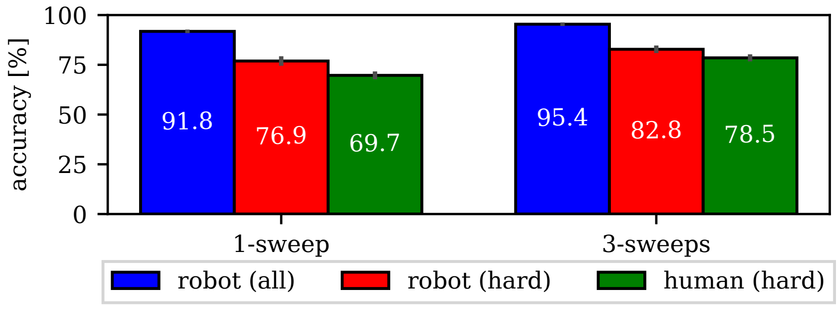


Figure 4.10: Human vs. robot material differentiation performance. The "hard" results are for the selected seven hardest material pairs, whereas the "all" result is for all possible material pairs.

4.6 Deep n-Shot Transfer Learning

In Fig. 4.2 n-shot transfer learning is summarized. The knowledge set $\widetilde{\mathcal{D}} = \{\widetilde{x}_i, \widetilde{y}_i\}_{i=1}^{\widetilde{N}}$ has many classes $\widetilde{C}$ and a large number of samples $\widetilde{n}$ per class, i.e., $\widetilde{N} = \widetilde{C}\widetilde{n}$. If the task set $\mathcal{D} = \{x_i, y_i\}_{i=1}^{N}$ consists of C classes with only few samples n per class, i.e., $N = Cn$, the problem is called a C-way n-shot learning problem. The goal in n-shot transfer learning is to reach high accuracy on test samples from the task classes, although the task set for training has only few samples. The trick is to transfer knowledge from the large knowledge set (with different classes).

In what follows, we first describe the three state of the art deep transfer learning methods we adopted for our tactile deep convolutional network TactNet-II and then we report the resulting performance for a 6-way n-shot learning task based on our 36 material dataset.

4.6.1 Fine-Tuning Method

As a typical CNN, our TactNet-II architecture (Fig. 4.11 a) consists of two parts: the automatic hierarchical feature extraction layers and the classification perceptron. The first part extracts important features from the input in a hierarchical fashion from low (elementary) to high semantic level [Xu et al., 2014, Krizhevsky et al., 2012], whereas the second part expands these features into a high dimensional space to fit a classification hyperplane.

The idea in fine-tuning is, inspired by [Hoffman et al., 2014], that the hierarchical features learned are generic when the number of training samples and classes is large enough. So, this features can be learned once from a large knowledge set and then be reused. For a new task, the input samples x are first transformed into this feature space, $u = \phi(x; \varphi)$, with the parameters φ previously learned from the knowledge set. Then, only the classification in the abstract feature space has to be learned from the (small) task set.

For a CNN, implementing this fine-tuning scheme means that all layers, except for the output layer (OL), are pre-trained on the knowledge set and then the weights are fixed. Only the OL is then trained on the task set. As a baseline, we also evaluate learning without knowledge transfer, i.e., all layers of the network are trained on the task set and the

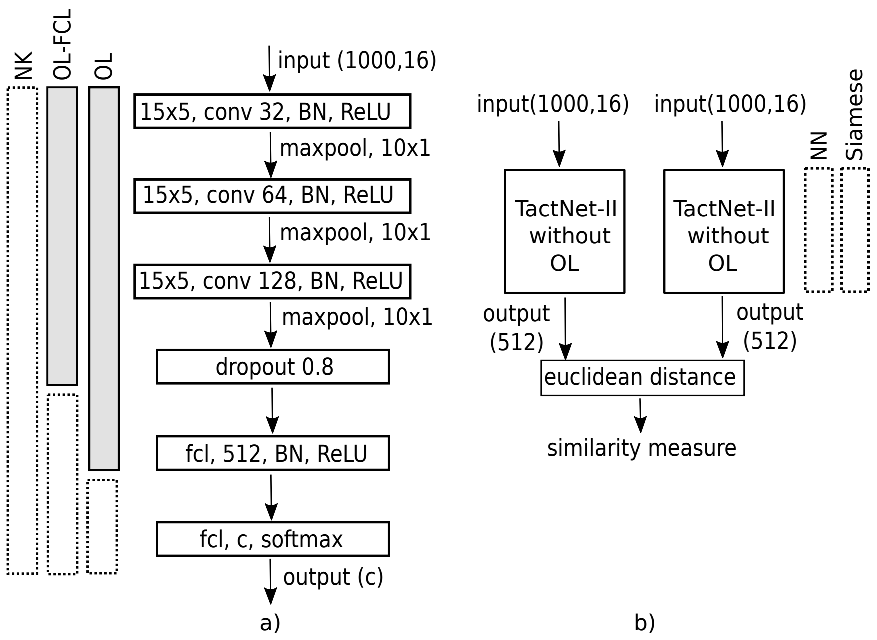


Figure 4.11: Deep transfer learning network architectures. a) The TactNet-II CNN. In transfer learning, first the full TactNet-II is trained on the knowledge set. For the *n*-shot learning, only parts of the network are re-trained. The boxes on the left indicate for the fine-tuning methods OL and OL-FCL which layers are kept fixed (gray) and which are re-trained (empty). As baseline, the NK method without knowledge transfer re-trains the full network. b) Matching network architecture. During testing, a pair of a test sample and a sample from the task set are mapped in parallel to the feature space by two trained identical TactNet-IIs. The Euclidean distance is then computed in this feature space. For the NN method, a standard pre-trained TactNet-II is used whereas the Siamese method trains the networks explicitly for discrimination between two given classes.

knowledge set is not used. We name this the no-pre-knowledge (NK) method. Fig. 4.11 a) gives a graphical summary of these two learning variants.

4.6.2 Matching Methods

Another method for reusing the learned feature mapping $u = \phi(x; \varphi)$, hence, the knowledge set, is to directly perform a nearest neighbor (NN) classification without any learning on the task set. The idea is that a simple distance metric in the abstract feature space can cover the class structure better than using the simple metric in the original sample space.

For 1-NN, given the task set $\mathcal{D} = \{x_i, y_i\}_{i=1}^N$ and a distance metric $d(u_1, u_2)$ in the feature space, a sample x is classified as y^* according to

$$i^* = \arg\min_i d\left(\phi(x; \varphi), \phi(x_i; \varphi)\right), \quad y^* = y_i^*.$$

We use the feature mapping of TactNet-II without the output layer and the simple Euclidian metric $d(u_1, u_2) = \|u_1 - u_2\|_2$. Fig. 4.11 b) depicts this *NN* method as a network structure, where two identical shortened TactNet-IIs are fed in parallel with x and the x_i.

An advanced variant of this matching idea is the *Siamese* network [KOCH et al., 2015]. In this method, also the training on the knowledge set for getting the feature mapping is done in the matching setup with the two identical twin networks (hence, the name Siamese). The idea is that this in way the mapping learns better features for the matching task than when trained for a classification task.

4.6.3 Concept Learning Method

In all methods described before, the transfer of knowledge is performed solely via the mapping into the feature space learned on the knowledge set. [BAUER et al., 2017] describe a simple probabilistic method that additionally transfers knowledge from the learned weights of the output layer in a OL fine-tuning setting like in Sec. 4.6.1.

Given the weights W of the C class neurons in the output layer, which have to be trained from the transformed task set samples $u = \phi(x; \varphi)$, with $x \in \mathcal{D}$ and $\widetilde{W}$ the weights of the $\widetilde{C}$ class neurons that have been trained on the knowledge set $\widetilde{\mathcal{D}}$. The full Bayesian approach in [BAUER et al., 2017] using the assumptions of a large number of classes $\widetilde{C}$ and samples per class $\widetilde{n}$ in the knowledge set and a small number of samples per class n in the task set, leads then to the following maximum a posteriori (MAP) point estimate for the predictive probability distribution for a new sample of a task class

$$\begin{aligned} &p(y|x, \mathcal{D}, \widetilde{\mathcal{D}}) \approx p(y|x, W_{\text{MAP}}), \\ &\text{with } W_{\text{MAP}} = \arg\max_W p(\mathcal{D}|W) p(W|\widetilde{W}). \end{aligned}$$

This is the same maximization expression as in standard MAP estimation but with a prior $p(W|\widetilde{W})$ for the weights W that depends on the weights $\widetilde{W}$ trained on the knowledge set.

With $W = (w_1, \dots, w_C)$ are the weights of the C task class neurons and $\widetilde{W} = (\widetilde{w}_1, \dots, \widetilde{w}_{\widetilde{C}})$ of the $\widetilde{C}$ knowledge class neurons and the probabilistic model assumption that the prior is a univariate Gaussian distribution, the prior can be written as

$$p(W|\widetilde{W}) = \prod_c^C \mathcal{N}(w_c|\mu_{\widetilde{w}}, \sigma^2_{\widetilde{w}}),$$
$$\mu_{\widetilde{w}} = \frac{1}{\widetilde{C}} \sum_c^{\widetilde{C}} \widetilde{w}_c, \;\; \sigma^2_{\widetilde{w}} = \frac{1}{\widetilde{C}} \sum_c^{\widetilde{C}} \frac{1}{F} (\widetilde{w}_c - \mu_{\widetilde{w}})^{\mathrm{T}} (\widetilde{w}_c - \mu_{\widetilde{w}}).$$

$\mu_{\widetilde{w}}$ and $\sigma^2_{\widetilde{w}}$ are the empirical mean and variance of the trained weights of the knowledge set class neurons and F is the dimensionality of the feature space (i.e., $F = \dim(u)$). It is important to note that the prior $p(W|\widetilde{W})$ is not centered at 0 but shifted by $\mu_{\widetilde{w}}$.

In summary, the concept learning method of [Bauer et al., 2017] biases the weights of the task class neurons to be close to the mean of the knowledge class neuron weights and the strength of this bias is inversely proportional to the variance of the knowledge class neuron weights.

For the error function, this prior results in a L_2 regularization term, but with the calculated optimal strength and the shift:

$$E(W) = -\log p(\mathcal{D}|W) + s \sum_c^C (w_c - m)^{\mathrm{T}} (w_c - m),$$
$$s = 1/(2\sigma^2_{\widetilde{w}}), \;\; m = \mu_{\widetilde{w}}.$$

We call the method using this regularization term *OL-mean*. The method OL from Sec 4.6.1 is the case where the strength s is set to the default, as used when training the TactNet-II.

4.6.4 Results

Cross-Validation

To evaluate the *n*-shot learning performance of a classifier with our material dataset with $K = 36$ classes, we use a double cross-validation scheme. On the class level, we split the $K = 36$ materials randomly into $C = 6$ task set classes and the $\widetilde{C} = 30$ knowledge set classes, i.e., we perform 6-way *n*-shot learning. If not mentioned otherwise, all results are reported for averaging over $S = 100$ random splits (but the same random splits are used for all learning methods). For each of theses material splits, we then perform another 5-fold cross-validation scheme to compute its mean accuracy.

Method Comparison

Fig. 4.12 shows that the *n*-shot transfer learning methods work very well and reach accuracies of $> 75\%$ even for 1-shot learning and $> 90\%$ for 10-shot learning. Compared to using no pre-knowledge (NK method), the accuracy gain is as high as 40%.

For 1-shot learning, the fine-tuning as well as the matching based NN methods perform comparably well, but for large n the accuracy of the NN method is almost 4% lower than

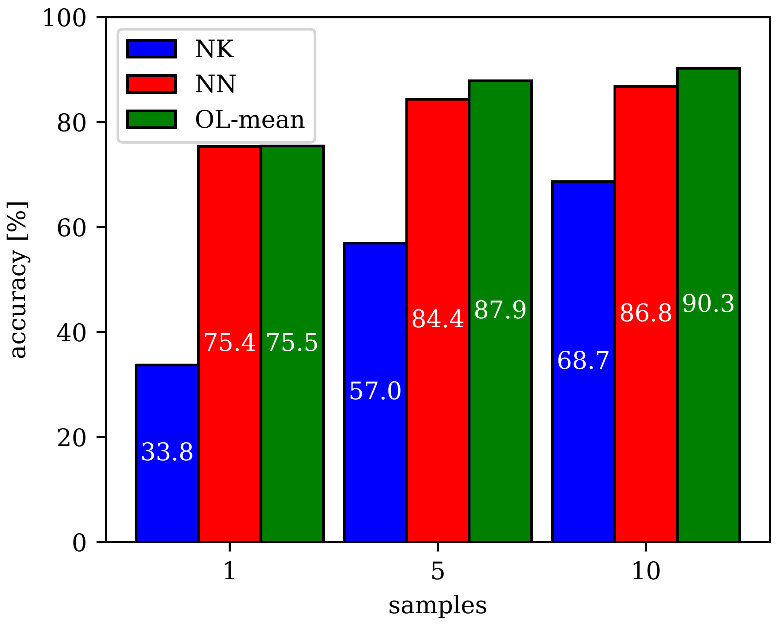


Figure 4.12: Performance comparison of the best transfer learning methods and the NK method without pre-knowledge.

OL-mean. This shows that learning of a shallow classifier in the feature space, although on only few examples, still covers better the class structure of the tactile data than a simple Euclidean metric.

The accuracy for the simple OL fine-tuning method is 75.0 % and, hence, 0.5 % lower than for OL-mean, proving that our adaption of the concept learning method to our TactNet-II network architecture works.

Feature Mapping

All the here used transfer learning methods depend on the assumption that TactNet-II learns a good feature mapping $u = \phi(x; \varphi)$ when trained on the knowledge set, so that classification in the feature space is easier than directly in the input space. To check this assumption, we perform t-SNE [VAN DER MAATEN and HINTON, 2008] dimensionality reduction in the input as well as in the learned feature space for the easy and hard to classify material combinations. Fig. 4.13 shows that in the input space the classes of the hard material combination are completely mixed up, while in the feature space, except for two classes, they are well separated. Also for the easy combination, the mapped samples are more clearly separated.

This visually underpins the rationale of the used deep transfer learning methods.

Sample Efficiency

In robotics, the usefulness of transfer learning depends on the reduction in the number of samples needed to reach a desired accuracy level. For this, Fig. 4.14 compares the accuracies of transfer learning (OL-mean) and learning without pre-knowledge (NK) for $n = 1, \ldots, 80$. For 1-shot learning, NK reaches the same accuracy level only for 15 times

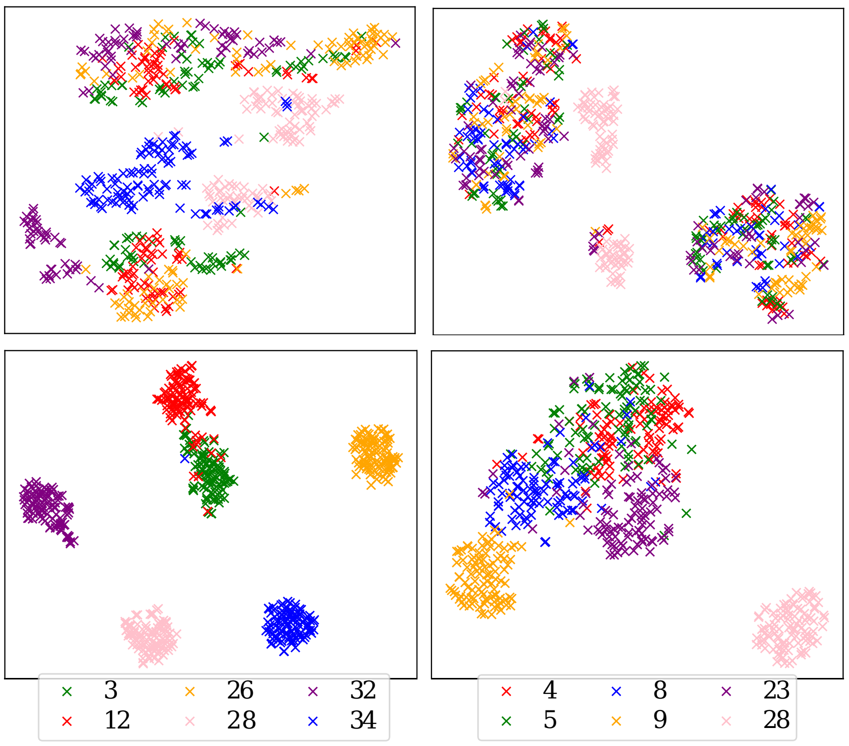


Figure 4.13: t-SNE on in the input (upper row) and learned feature space (lower row) for the easy (left column) and hard material combinations (right column).

more samples. In case an accuracy of $> 90\%$ is desired, 10-shot transfer learning still needs about 5 times less samples than NK.

4.7 Summary

In [17], we have shown for the first time that a robot equipped with a flexible tactile skin can exceed human performance in tactile material classification. Tactile material classification is a rather high-level task as it involves longterm memory, but also for the low-level task of tactile material differentiation, the robot performs better than the human, hence, the robot truly reaches *superhuman* tactile performance.

This result was achieved in a real world, non-benchmark setup with a commercially available flexible tactile skin that was simply taped to Agile Justin's soft fingertip. With this setup, we recorded a new tactile dataset with 3600 samples from 36 everyday household materials and made it publicly available at [TULBURE and BÄUML, 2018].

To process the complex and noisy spatio-temporal signal of the tactile skin, we designed and implemented a deep convolutional network architecture for tactile material classification, TactNet-II, using directly the raw 16000 dimensional spatio-temporal signal as input. For training, we adapted state of the art deep learning methods like adversarial training and efficient Bayesian deep learning (MC dropout) for TactNet-II and reached an accuracy for the material classification task as high as 95.0 %. For material differentiation

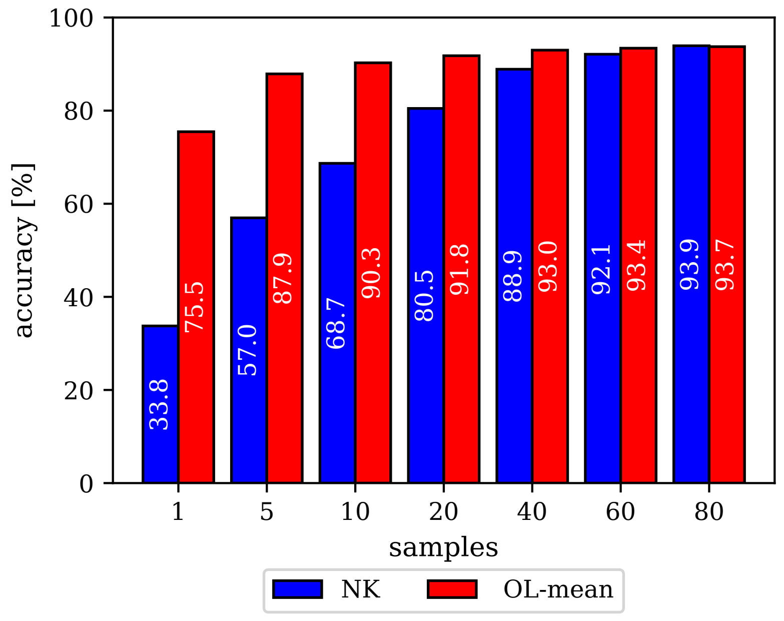


Figure 4.14: Performance comparison of n-shot transfer learning (OL-mean method) and learning without prior knowledge (NK).

we used TactNet-II as a building block in a Siamese-like network architecture and reached an accuracy as high as 95.4 %.

TactNet-II is an enhanced version of TactNet from our previous work on tactile material classification [4] where we used a smaller dataset of 6 materials. In [4] we also compared TactNet with classical tactile classification methods which are based on low dimensional, manually designed features. In Sec. 4.4.2 we have updated this comparison to the new TactNet-II and the full 36 material dataset. The results show that the classical methods perform poorly on the complex spatio-temporal signal of the flexible tactile skin which clearly shows the superiority of automatic feature extraction in deep end-to-end learning.

As a baseline for highly developed tactile sensing, we performed a thorough human performance experiment with 15 subjects in [17]. In the material classification task, the humans performed poorly with an accuracy of at least 30% worse than the robot. For the low-level material differentiation task, the human performance was still significantly lower than the robot performance, but by a smaller margin.

Sample efficiency is key in making learning feasible in robotics, as collecting data is an active and therefore expensive task. In [21] we have shown for the first time, that deep end-to-end transfer learning for tactile material classification is feasible. By adapting state of the art deep transfer learning methods to our TactNet-II deep CNN architecture, we reached a classification accuracy as high as 75.5% for 6-way 1-shot learning and of over 90% for 10-shot learning. This leads to a 15 and 5 time reduction in the number of samples needed compared to learning without knowledge transfer.

Our analysis shows that the probabilistic concept learning method [Bauer et al., 2017], which has so far only been used in a vision benchmark example, can be applied to the real world material classification task. By using the TactNet-II, it outperforms all other tested deep n-shot transfer learning methods for all n.

This work on deep learning based tactile material classification resulted in a double award finalist [4] at the major robotic conference IROS 2018 and international TV coverage and was robustly demonstrated live at public institute events and numerous lab demonstrations.

In the future, we want to apply deep learning based processing of the spatio-temporal tactile sensor signal for tasks like slippage detection in grasping or the evaluation of the stability of a grasp on an object.

Chapter 5

Conclusion

In this thesis, the versatility of the advanced torque controlled mobile humanoid robot Agile Justin has been significantly increased into two important directions. First, although originally developed as a research platform in dextrous manipulation, Agile Justin can now also execute complex dynamic manipulation tasks. This was demonstrated with the challenging task of catching up to two simultaneously thrown balls with its hands. Second, Agile Justin has now highly developed tactile sensing capabilities that are important for dextrous fine manipulation. We have demonstrated its tactile capabilities with the delicate task of identifying an objects material simply by gently sweeping with a fingertip over its surface.

The hard realtime capable and highly performant communication layer of our aRDx robotic software framework is key for realizing complex dynamic manipulation tasks. It is generally accepted that only by using a component based system architecture in the development of mobile manipulation applications one can handle the complexity and the computational demands (including parallel and distributed execution) of their deep sensor-perception-planning-action loops. But only due to aRDx's performance, such a component based system architecture can also be used for dynamic manipulation tasks with their tight timing constraints.

Besides hard realtime determinism, aRDx provides optimal data packet transport, i.e., zero-copy semantics in the process and host and copy-once semantics in the distributed domain, as well as advanced channel synchronization. It has an elegant hierarchical architecture where, starting from a simple channel mechanism for intra-process packet transport, the communication in the process, host and distributed domain build upon each other.

In implementing the challenging dynamic ball catching application for Agile Justin, we took heavy use of all the features of aRDx. aRDx is not only used for the communication between the individual subsystems but inside the subsystems as well, e.g., for parallel execution or synchronization of sensor inputs. Besides developing the challenging visual ball tracking using only onboard sensing while "everything is moving" and an automatic and self-contained calibration procedure for the multi-sensorial upper body to provide the necessary precision, a major contribution is the realization of a unified generation of the reaching motion for the arms. The catch point selection, motion planning obeying dynamic and geometrical constraints and the joint interpolation are subsumed in one nonlinear constrained optimization problem which is solved in realtime. Agile Justin reaches a success rate of over 90 % in catching up to two simultaneously thrown balls and the optimization-based planning approach allows for the realization of different catch behaviors.

We used a similar optimization-based approach for planning an optimal whole-body trajectory, including the mobile platform, to make Agile Justin throwing the ball back.

Another important component of Agile Justin, but in this thesis only briefly mentioned, is the realtime visual 3D modeling of the environment which is executed on a wirelessly coupled GPU cluster via aRDx communication. On the resulting dense and high-resolution 3D models we perform optimization-based motion planning for autonomously acting in previously unknown environments.

For the highly sensitive task of tactile material classification, the signal of a flexible pressure-sensitive skin which is simply taped to Agile Justin's soft fingertip is used. We presented our deep convolutional network architecture TactNet-II which directly uses the raw 16000 dimensional complex and noisy spatio-temporal signal from one sweep over a material as input. TactNet-II is trained with advanced deep learning methods like adversarial training and efficient Bayesian deep learning, namely Monte Carlo dropout, and reaches a classification accuracy of over 86 %.

For comparison, we also performed a thorough human performance experiment with 15 subjects using the very same 36 material. In the the more high-level cognitive material classification task, the humans performed poorly with an accuracy at least 30% worse than the robot. But also for the low-level material differentiation task, the human performance was still significantly lower than the robot performance, but by a smaller margin.

Sample efficiency is key in making learning feasible in robotics, as collecting data is an active and, hence, expensive task. We have shown for the first time, that deep end-to-end transfer learning for tactile material classification is feasible. By adapting state of the art deep transfer learning methods to our TactNet-II deep CNN architecture, we reached a classification accuracy in this real world setting as high as 75.5% for 6-way 1-shot learning and of over 90% for 10-shot learning. This leads to a 15 and 5 time reduction in the number of samples needed compared to learning without knowledge transfer.

The presented work gained respectable attention with the publications becoming once an award winner and five times award finalists at major robotic conferences. In addition, Agile Justin has been covered many times in international media including more than 500000 overall clicks on YouTube. But the new capabilities of Agile Justin also work robustly and could be regularly shown at trade fairs (Automatica 2010 - 2014), public institute events and numerous lab demonstrations.

In summary, as stated in [ACKERMAN, 2014], Agile Justin is "arguably one of the most, if not the most, capable dual-armed mobile humanoid robots in existence". Regarding its mechatronic and sensor as well as its perception and planning capabilities, Agile Justin already comes close to human versatility – at least in a lab setting. Hence, Agile Justin is an almost ideal research platform for intelligent mobile manipulation where advances in artificial intelligence are no longer hindered by the constraints of the robotic system. The approach we are currently pursuing is to use autonomous learning, especially deep reinforcement learning, as the core principle in autonomous humanoid robots which robustly and dextrously operate in complex and changing environments.

List of Publications by the Author

Reviewed Publications

In Scientific Journals

[1] BÄUML, BERTHOLD and GERD HIRZINGER: *When hard realtime matters: Software for complex mechatronic systems*. Robotics and Autonomous Systems, 56(1):5–13, 2008.

My share is 95%.
I analyzed the demands of software for advanced mechatronical systems, designed and implemented the software framework aRD, the predecessor of aRDx, and performed benchmark experiments.

[2] BIRBACH, OLIVER, UDO FRESE and BERTHOLD BÄUML: *Rapid Calibration of a Multi-Sensorial Humanoid's Upper Body: An Automatic and Self-Contained Approach*. International Journal of Robotics Research, 2014.

My share is 25%.
This paper is an updated and extended version of [10]. The main extensions are a realtime and parallel calibration pipeline, which I designed based on aRDx, and new experiments, which I co-conducted. I wrote parts of the paper.

[3] HAMMER, TOBIAS and BERTHOLD BÄUML: *The Communication Layer of the aRDx Software Framework: Highly Performant and Realtime Deterministic*. Journal of Intelligent and Robotic Systems, 77, 2015.

My share is 50%.
This paper is an updated and extended version of [13] and describes the application of aRDx for the system architecture of Agile Justin which I designed.

At Peer-Reviewed Conferences

[4] BAISHYA, SHIV and BERTHOLD BÄUML: *Robust Material Classification with a Tactile Skin Using Deep Learning*. In *Proc. IEEE International Conference on Intelligent Robots and Systems*. **Best Cognitive Robotics Paper Finalist.**
Best Student Paper Finalist., 2016.

My share is 49%.
This publication resulted from the first author's master's thesis that I initiated and advised. I designed the feature sets and classification methods for the classical approach to material classification. In addition, I designed the tactile deep convolutional network architecture TactNet and the experimental setup for the performance evaluation. The main parts of the paper, including the result discussion, were written by me.

[5] BÄUML, B., T. HAMMER, R. WAGNER, O. BIRBACH, TH. GUMPERT, F. ZHI, U. HILLENBRAND, ST. BEER, W. FRIEDL and J. BUTTERFASS: *Agile Justin: An Upgraded Member of DLR's Family of Lightweight and Torque Controlled Humanoids*. In *Proc. IEEE International Conference on Robotics and Automation*. **Best Video Award.**, 2014.

My share is 50%.
I designed the holistic system architecture of Agile Justin. In addition, I developed and implemented the optimization-based ball throwing, designed and implemented the optimization based path planning based on self-acquired 3D models, developed and implemented the tactile material classification and co-developed the auto-calibration method for the multi-sensorial upper body.The aRDx robotic software framework was designed and co-implemented by me. I directed the video and wrote the accompanying short paper.

[6] BÄUML, BERTHOLD, OLIVER BIRBACH, THOMAS WIMBÖCK, UDO FRESE, ALEXANDER DIETRICH and GERD HIRZINGER: *Catching Flying Balls with a Mobile Humanoid: System Overview and Design Considerations*. In *Proc. IEEE-RAS International Conference on Humanoid Robots*, 2011.

My share is 60%.
I analyzed the requirements for the holistic system architecture and designed it. Based on the aRD and aRDx frameworks, I implemented the mapping of the software components onto the robot's distributed hardware resources. In addition, I contributed the physical ball trajectory model for the visual ball tracking. I developed and implemented the realtime generation of the reaching motion based on my optimization-based planner for the arms and a smart way for using the robot's kinematic sub-chains. The experiments were designed and co-conducted by me.

[7] BÄUML, BERTHOLD, FLORIAN SCHMIDT, THOMAS WIMBÖCK, OLIVER BIRBACH, ALEX DIETRICH, MATTHIAS FUCHS, WERNER FRIEDL, UDO FRESE, CHRISTOPH BORST, MARKUS GREBENSTEIN, OLIVER EIBERGER and GERD HIRZINGER: *Catching Flying Balls and Preparing Coffee: Humanoid Rollin'Justin Performs Dynamic and Sensitive Tasks*. In *Proc. IEEE International Conference on Robotics and Automation*. **Best Video Finalist.**, 2011.

My share is 30%.
I directed and contributed to the first part of the video (70% of the video) presenting ball catching on the mobile humanoid robot Justin and wrote the accompanying short paper. In addition, I co-developed the visual ball tracking and developed and implemented the optimization-based realtime generation of the reaching motion. I designed the holistic system architecture and, based on the aRD and aRDx framework, implemented the mapping of the software components onto the distributed hardware resources of Agile Justin. The experiments were co-conducted by me.

[8] BÄUML, BERTHOLD, THOMAS WIMBÖCK and GERD HIRZINGER: *Kinematically Optimal Catching a Flying Ball with a Hand-Arm-System*. In *Proc. IEEE International Conference on Intelligent Robots and Systems*, 2010.

My share is 80%.
I developed the generation of the reaching motion of the arm as a unified nonlinear constraint

optimization problem and implemented a parallel distributed realtime solver. In addition, I designed and implemented the system architecture fulfilling the realtime demands of the dynamic task of ball catching based on our aRD software framework. The validation experiments were conducted and analyzed by me and I wrote the paper.

[9] BIRBACH, OLIVER and BERTHOLD BÄUML: *Calibrating a Pair of Inertial Sensors at Opposite Ends of an Imperfect Kinematic Chain*. In *Proc. IEEE International Conference on Intelligent Robots and Systems*, 2014.

My share is 30%.
I provided the idea of how to calibrate the two IMUs of Agile Justin despite the imperfect kinematic chain between them. In addition, I co-developed the mathematical models and conducted the experiments on Agile Justin.

[10] BIRBACH, OLIVER, BERTHOLD BÄUML and UDO FRESE: *Automatic and Self-Contained Calibration of a Multi-Sensorial Humanoid's Upper Body*. In *Proc. IEEE International Conference on Robotics and Automation*. **Best Vision Paper Finalist.**, 2012.

My share is 20%.
I contributed the underlying idea for automatic calibration to use the robot to automatically record many, albeit relatively imprecise, measurements instead of a few measurements of a precise but external calibration pattern. In addition, I co-developed the models for the sensors and the kinematic chain with soft joints and integrated the auto-calibration implementation on Agile Justin using our aRDx software framework and co-conducted the experiments.

[11] BIRBACH, OLIVER, UDO FRESE and BERTHOLD BÄUML: *Realtime Perception for Catching a Flying Ball with a Mobile Humanoid*. In *IEEE Proc. International Conference on Robotics and Automation*, 2011.

My share is 20%.
I contributed the physical ball trajectory model and co-developed the sensor models. In addition, I co-invented the idea of using a normalization scheme based on the local image intensity variance for robust circle detection. The integration of the ball tracking onto the distributed system resources of Agile Justin based on aRD/aRDx was designed and co-implemented by me. I also co-conducted the evaluation experiments.

[12] CARRILLO, HENRY, OLIVER BIRBACH, HOLGER TÄUBIG, BERTHOLD BÄUML, UDO FRESE and JOSÉ A. CASTELLANOS: *On Task-Oriented Criteria for Configurations Selection in Robot Calibration*. In *Proc. IEEE International Conference on Robotics and Automation*, 2013.

My share is 10%.
This is based on our prior contributions to automatic humanoid robot calibration. I assisted during the design of the proposed method and co-conducted the experiments on the robot for evaluation.

[13] HAMMER, TOBIAS and BERTHOLD BÄUML: *The Highly Performant and Realtime Deterministic Communication Layer of the aRDx Software Framework*. In *Proc. Int. Conf. on Advanced Robotics (ICAR)*, 2013.

My share is 50%.
I provided the requirement analysis and design considerations for a performant robotic software framework and designed the aRDx architecture and its elegant hierarchical implementation. In addition, I designed the stress test benchmarks for the communication performance of robotic frameworks and did the discussion of the results and wrote the main part of the paper.

[14] KONIETSCHKE, RAINER, TIM BODENMUELLER, CHRISTIAN RINK, ANDREA SCHWIER, BERTHOLD BAEUML and GERD HIRZINGER: *Optimal Setup of the DLR MiroSurge Telerobotic System for Minimally Invasive Surgery*. In *Proc. IEEE International Conference Robotics and Automation (ICRA)*, 2011.

My share is 10%.
I implemented the GUI on the handheld device (iPhone) and integrated it in the aRD based robotic surgery system architecture. For the latter, I ported our aRD framework to the iOS operating system.

[15] TÄUBIG, H., B. BÄUML and U. FRESE: *Real-Time Swept Volume and Distance Computation for Self Collision Detection*. In *Proc. IEEE International Conference on Intelligent Robots and Systems*. **Best Student Paper Finalist.**, 2011.

My share is 15%.
I contributed to the design and the precise and clear writing up of the method in the paper. Based on our aRD framework, I designed and co-implemented the integration of the realtime collision detection on Agile Justin and conducted the experiments.

[16] TÄUBIG, HOLGER, BERTHOLD BÄUML and UDO FRESE: *Real-time Continuous Collision Detection for Mobile Manipulators – A General Approach*. In *Proc. IEEE-RAS International Conference on Humanoid Robots*, 2012.

My share is 10%.
I contributed to the design and the clear formulation of the method. Based on our aRD framework, I designed and co-implemented the integration of the realtime collision detection on Agile Justin and conducted the experiments.

[17] TULBURE, ANDREEA and BERTHOLD BÄUML: *Superhuman Performance in Tactile Material Classification and Differentiation with a Flexible Pressure-Sensitive Skin*. In *Proc. IEEE/RAS International Conference on Humanoid Robots*, 2018.

My share is 50%.
This publication resulted from the first author's master's thesis that I initiated and advised. I designed the extended tactile deep convolutional network architecture TactNet-II and the experimental scenarios of material classification and differentiation. In addition, I designed the human performance experiments and wrote main parts of the paper.

[18] WAGNER, RENÉ, BERTHOLD BÄUML and UDO FRESE: *3D Modeling, Distance and Gradient Computation for Motion Planning: A Direct GPGPU Approach*. In *Proc. IEEE International Conference on Robotics and Automation*, 2013.

My share is 20%.
I contributed the underlying idea of efficiently combining the models from dense SLAM with

optimization-based path planning (OMP) by directly computing the EDT (Euclidian distance transform) from the TSDF models. I implemented the optimization-based path planning, designed the aRDx based software architecture for integrating the method on Agile Justin and co-conducted the experiments. The part of the paper about OMP were written by me.

[19] WAGNER, RENE, UDO FRESE and BERTHOLD BÄUML: *Real-Time Dense Multi-Scale Workspace Modeling on a Humanoid Robot*. In *Proc. IEEE International Conference on Intelligent Robots and Systems*, 2013.

My share is 15%.
I co-designed the multi-scale approach including the computation of the EDT for path planning and co-conducted the experiments on Agile Justin and wrote parts of the paper.

At Peer-Reviewed Workshops

[20] BÄUML, BERTHOLD: *One for (Almost) All: Using a Modern Programmable Programming Language in Robotics*. In *Proc. IEEE International Conference on Robotics and Automation, Keynote of SDIR Workshop*, 2013.

My share is 100%.
This paper provides an analysis of using the Racket programming language as the base for the higher level layers of our aRDx robotic software framework.

Submitted Publications

At Peer-Reviewed Conferences

[21] BÄUML, BERTHOLD and ANDREEA TULBURE: *Deep n-Shot Transfer Learning for Tactile Material Classification with a Flexible Pressure-Sensitive Skin*. In *Proc. IEEE International Conference on Robotics and Automation*, 2019.

My share is 70%.
I designed the adaption of state of the art n-shot transfer learning methods to our tactile deep convolutional network architecture TactNet-II. In addition, I designed the evaluation scenarios based on splitting our material dataset in a knowledge and task set and provided the discussion of the results. The paper was written by me.

References

[Abadi et al., 2015] Abadi, Martin, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu and X. Zheng (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. Technical Report, Google Research.

[Ackerman, 2014] Ackerman, Evan (2014). *IEEE Spectrum Robotics Video Monday*. `https://spectrum.ieee.org/automaton/robotics/robotics-hardware/video-monday-icra-2014`.

[ADLINK] ADLINK. *ADLINK OpenSplice DDS*. `http://www.prismtech.com/dds-community`.

[Asfour et al., 2013] Asfour, T., J. Schill, H. Peters, C. Klas, J. Bückera, C. Sander, S. Schulz, A. Kargov, T. Werner, and V. Bartenbach (2013). *ARMAR-4: A 63 DOF Torque Controlled Humanoid Robotnoid Robot*. In *Proc. IEEE/RAS International Conference on Humanoid Robots*.

[Asfour et al., 2018] Asfour, Tamim, L. Kaul, M. Wächter, S. Ottenhaus, P. Weiner, S. Rader, R. Grimm, Y. Zhou, M. Grotz, F. Paus, D. Shingarey and H. Haubert (2018). *ARMAR-6: A Collaborative Humanoid Robot for Industrial Environments*. In *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*.

[Bae et al., 2012] Bae, J., S. Park, J. Park, M. Baeg, D. Kim and S. Oh (2012). *Development of a low cost anthropomorphic robot hand with high capability*. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4776–4782.

[Baishya and Bäuml, 2016] Baishya, Shiv and B. Bäuml (2016). *Robust Material Classification with a Tactile Skin Using Deep Learning*. `https://www.youtube.com/watch?v=623yRPx9Pkc`.

[Bauer et al., 2017] Bauer, Matthias, M. Rojas-Carulla, J. Swiatkowski, B. Schoelkopf and R. E. Turner (2017). *Discriminative k-shot learning using probabilistic models*. In *NIPS workshop on Bayesian Deep Learning*.

[Bäuml, 2012] Bäuml, Berthold (2012). *Agile and Rollin' Justin Playing Ball*. `http://www.youtube.com/watch?v=93WHRSKg3gE`.

[Bäuml and Hirzinger, 2006] Bäuml, Berthold and G. Hirzinger (2006). *Agile Robot Development (aRD): A Pragmatic Approach to Robotic Software*. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[Birbach, 2012] Birbach, Oliver (2012). *Tracking and Calibration for a Ball Catching Humanoid Robot*. PhD thesis, Universität Bremen, Fachbereich 3 (Mathematik und Informatik).

[BIRBACH and FRESE, 2013] BIRBACH, OLIVER and U. FRESE (2013). *A Precise Tracking Algorithm Based on Raw Detector Responses and a Physical Motion Model*. In *Proc. IEEE International Conference on Robotics and Automation*, pp. 4746 – 4751.

[BISHOP, 2006] BISHOP, CHRISTOPHER M. (2006). *Pattern Recognition and Machine Learning*. Springer.

[BORST et al., 2009] BORST, C., T. WIMBÖCK, F. SCHMIDT, M. FUCHS, B. BRUNNER, F. ZACHARIAS, P. R. GIORDANO, R. KONIETSCHKE, W. SEPP, S. FUCHS, C. RINK, A. ALBU-SCHÄFFER and G. HIRZINGER (2009). *Rollin' Justin: Mobile Platform with Variable Base*. In *Proc. IEEE International Conference on Robotics and Automation*, pp. 1597–1598.

[BORST et al., 2007] BORST, CH., C. OTT, T. WIMBOECK, B. BRUNNER, F. ZACHARIAS, B. BÄUML, U. HILLENBRAND, S. HADDADIN, A. ALBU-SCHAEFFER and G. HIRZINGER (2007). *A Humanoid Upper Body System for Two-Handed Manipulation*. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*.

[BOSTON DYNAMICS, 2018] BOSTON DYNAMICS (2018). *Atlas: The World's Most Dynamic Humanoid*. `https://www.bostondynamics.com/atlas`.

[BUTTERFASS et al., 2001] BUTTERFASS, J., M. GREBENSTEIN, H. LIU and G. HIRZINGER (2001). *DLR-Hand II: Next Generation of a Dextrous Robot Hand*. In *Proc. IEEE International Conference on Robotics and Automation*, pp. 109–114.

[BYRAVAN et al., 2014] BYRAVAN, A., B. BOOTS, S. SRINIVASA and D. FOX (2014). *Space-Time Functional Gradient Optimization for Motion Planning*. In *Proc IEEE International Conference on Robotics and Automation*.

[CAP'N PROTO] CAP'N PROTO. *Cap'n Proto*. `https://capnproto.org`.

[CHATHURANGA et al., 2015] CHATHURANGA, DAMITH SURESH, Z. WANG, Y. NOH, T. NANAYAKKARA and S. HIRAI (2015). *Robust Real time Material Classification Algorithm Using Soft Three Axis Tactile Sensor: Evaluation of the Algorithm*. In *Proc. IEEE International Conference on Intelligent Robots and Systems*.

[CHENG et al., 2006] CHENG, G., S.-H. HYON, J. MORIMOTO, A. UDE, G. COLVIN, W. SCROGGIN and S. C. JACOBSEN (2006). *Cb: A humanoid research platform for exploring neuroscience*. IEEE-RAS International Conference on Humanoid Robots.

[COHEN, 1995] COHEN, LEON (1995). *Time-Frequency Analysis*. Prentice-Hall.

[CONNEXT, 2018] CONNEXT, RTI (2018). *Connext DDS Micro*. `https://www.rti.com/products/connext-dds-micro`.

[COX and HINGORANI, 1996] COX, INGEMAR J. and S. L. HINGORANI (1996). *An Efficient Implementation of Reid's Multiple Hypothesis Tracking Algorithm and Its Evaluation for the Purpose of Visual Tracking*. IEEE Trans. Pattern Anal. Mach. Intell., 18(2):138–150.

[DAHIYA et al., 2013] DAHIYA, RAVINDER S., P. MITTENDORFER, M. VALLE, G. CHENG and V. J. LUMELSKY (2013). *Directions Toward Effective Utilization of Tactile Skin: A Review*. IEEE Sensors Journal, 13(11).

[EGUILUZ et al., 2016] EGUILUZ, A. GOMEZ, I. RANO, S. COLEMAN and T. MCGINNITY (2016). *Continuous Material Identification through Tactile Sensing*. In *Proc. Int. Joint Conference on Neural Networks*.

[EINHORN et al., 2012] EINHORN, ERIK, T. LANGNER, R. STRICKER, C. MARTIN and H.-M. GROSS (2012). *MIRA - Middleware for Robotic Applications.* In *Proc. IEEE International Conference on Intelligent Robots and Systems.*

[ENGLSBERGER et al., 2014] ENGLSBERGER, J., A. WERNER, C. OTT, B. HENZE, M. A. ROA, G. GAROFALO, R. BURGER, A. BEYER, O. EIBERGER, K. SCHMID and A. ALBU-SCHÄFFER (2014). *Overview of the torque-controlled humanoid robot TORO.* In *2014 IEEE-RAS International Conference on Humanoid Robots*, pp. 916–923.

[EPROSIMA] EPROSIMA. *eProsima FastRTPS.* `http://www.eprosima.com/index.php/products-all/eprosima-fast-rtps`.

[FEI-FEI et al., 2006] FEI-FEI, LI, R. FERGUS and P. PERONA (2006). *One-shot learning of object categories.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 28(4):594–611.

[FISHEL and LOEB, 2012] FISHEL, J.A. and G. LOEB (2012). *Bayesian exploration for intelligent identification of textures.* Frontiers in Neurorobotics, 6(4):1–20.

[FLATBUFFERS] FLATBUFFERS. *FlatBuffers.* `https://google.github.io/flatbuffers/`.

[FRESE et al., 2001] FRESE, U., B. BÄUML, S. HAIDACHER, G. SCHREIBER, I. SCHAEFER, M. HÄHNLE and G. HIRZINGER (2001). *Off-the-Shelf Vision for a Robotic Ball Catcher.* In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems.*

[FUCHS et al., 2010] FUCHS, STEFAN, S. HADDADIN, M. KELLER, S. PARUSEL, A. KOLB and M. SUPPA (2010). *Cooperative bin-picking with Time-of-Flight camera and impedance controlled DLR lightweight robot III.* In *Proc. IEEE International Conference on Intelligent Robots and Systems.*

[GAL and GHAHRAMANI, 2016] GAL, YARIN and Z. GHAHRAMANI (2016). *Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning.* In *Proc. Int. Conference on Machine Learning.*

[GALLMEISTER, 1995] GALLMEISTER, BILL O. (1995). *POSIX. 4: Programming for the Real World.* O'Reilly.

[GAO et al., 2016] GAO, YANG, L. A. HENDRICKS, K. J. KUCHENBECKER and T. DARRELL (2016). *Deep Learning for Tactile Understanding From Visual and Haptic Data.* In *Proc. IEEE International Conference on Robotics and Automation.*

[GERKEY, 2015] GERKEY, BRIAN (2015). *Why ROS 2.0?.* `https://design.ros2.org/articles/why_ros2.html`.

[GERKEY, 2018] GERKEY, BRIAN (2018). *Introducing the ROS 2 Technical Steering Committee.* `https://discourse.ros.org/t/introducing-the-ros-2-technical-steering-committee/6132`.

[GNU] GNU. *GCC, the GNU Compiler Collection.* `gnugcc`.

[GONZALEZ-BANOS et al., 2006] GONZALEZ-BANOS, H.H., D. HSU and J. LATOMBE (2006). *Motion planning: Recent developments.* In GE, S.S. UND F.L. LEWIS, ed.: *Automous Mobile Robots: Sensing, Control, Decision-Making and Applications. CRC.*

[GOODFELLOW et al., 2014] GOODFELLOW, I. J., J. SHLENS and C. SZEGEDY (2014). *Explaining and Harnessing Adversarial Examples.* ArXiv e-prints.

[GUTIÉRREZ et al., 2018] GUTIÉRREZ, CARLOS SAN VICENTE, L. U. S. JUAN, I. Z. UGARTE and V. M. VILCHES (2018). *Towards a distributed and real-time framework for robots: Evaluation of ROS 2.0 communications for real-time robotic applications*.

[HAGN et al., 2010] HAGN, ULRICH, R. KONIETSCHKE, A. TOBERGTE, M. NICKL, S. JÖRG, B. KÜBLER, G. PASSIG, M. GRÖGER, F. FRÖHLICH, U. SEIBOLD, L. LE-TIEN, A. ALBU-SCHÄFFER, A. NOTHHELFER, F. HACKER, M. GREBENSTEIN and G. HIRZINGER (2010). *DLR MiroSurge: a versatile system for research in endoscopic telesurgery*. Int J Comput Assist Radiol Surg, 5(2):183–193.

[HENNING, 2004] HENNING, M. (2004). *A new approach to object-oriented middleware*. IEEE Internet Computing, 8(1):66–75.

[HINTON et al., 2012] HINTON, GEOFFREY, L. DENG, D. YU, G. DAHL, A. MOHAMED, N. JAITLY, A. SENIOR, V. VANHOUCKE, P. NGUYEN, T. SAINATH and B. KINGSBURY (2012). *Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups*. IEEE Signal Processing Magazine, 29(6):82—97.

[HIRZINGER et al., 2002] HIRZINGER, G., N. SPORER, A. ALBU-SCHÄFFER, M. HÄHNLE, R. KRENN, A. PASCUCCI and M. SCHEDL (2002). *DLR's Torque-Controlled Light Weight Robot III - are we Reaching the Technological Limits now?*. In *Proc. IEEE International Conference on Robotics and Automation*, pp. 1710–1716.

[HOELSCHER et al., 2015] HOELSCHER, JANINE, J. PETERS and T. HERMANS (2015). *Evaluation of Tactile Feature Extraction for Interactive Object Recognition*. In *Proc. IEEE-RAS International Conference on Humanoid Robots*.

[HOFFMAN et al., 2014] HOFFMAN, JUDY, E. TZENG, J. DONAHUE, Y. JIA, K. SAENKO and T. DARRELL (2014). *One-Shot Adaptation of Supervised Deep Convolutional Models*. In *Proc. International Conference in Learning and Representation*.

[HONG and SLOTINE, 1995] HONG, W. and J. SLOTINE (1995). *Experiments in Hand-Eye Coordination Using Active Vision*. In *Proc. Fourth International Symposium on Experimental Robotics*.

[HÖPPNER et al., 2017] HÖPPNER, HANNES, M. GROSSE-DUNKER, G. STILLFRIED, J. BAYER and P. VAN DER SMAGT (2017). *Key Insights into Hand Biomechanics: Human Grip Stiffness Can Be Decoupled from Force by Cocontraction and Predicted from Electromyography*. Frontiers in Neurorobotics, 11:17.

[HOVE and SLOTINE, 1991] HOVE, B.M. and J. SLOTINE (1991). *Experiments in Robotic Catching*. In *Proc. IEEE American Control Conference*, pp. 380–385.

[IOFFE and SZEGEDY, 2015] IOFFE, SERGEY and C. SZEGEDY (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. In *ICML*.

[IWATA and SUGANO, 2009] IWATA, HIROYASU and S. SUGANO (2009). *Design of Human Symbiotic Robot TWENDY-ONE*. In *Proc. IEEE International Conference on Robotics and Automation*.

[JÖRG et al., 2014] JÖRG, STEFAN, J. TULLY and A. ALBU-SCHÄFFER (2014). *The Hardware Abstraction Layer - Supporting Control Design by Tackling the Complexity of Humanoid Robot Hardware*. In *Proc. IEEE International Conference on Robotics and Automation*.

[KABOLI et al., 2014] KABOLI, MOHSEN, P. MITTENDORFER, V. HUGEL and G. CHENG (2014). *Humanoids Learn Object Properties From Robust Tactile Feature Descriptors via Multi-Modal Artificial Skin*. In *Proc. IEEE/RAS International Conference on Humanoid Robots*.

[KABOLI et al., 2015] KABOLI, MOHSEN, A. D. L. ROSA, R. WALKER and G. CHENG (2015). *In-Hand Object Recognition via Texture Properties with Robotic Hands, Artificial Skin, and Novel Tactile Descriptors*. In *Proc. IEEE/RAS International Conference on Humanoid Robots*.

[KABOLI et al., 2016] KABOLI, MOHSEN, R. WALKER and G. CHENG (2016). *Re-using Prior Tactile Experience by Robotic Hands to Discriminate In-Hand Objects via Texture Properties*. In *Proc. IEEE International Conference on Robotics and Automation*.

[KANEKO et al., 2011] KANEKO, K., F. KANEHIRO, M. MORISAWA, K. AKACHI, G. MIYAMORI, A. HAYASHI and N. KANEHIRA (2011). *Humanoid robot HRP-4 - humanoid robotics platform with lightweight and slim body*. In *Proc. IEEE International Conference on Intelligent Robots and System*.

[KAPPASSOV et al., 2015] KAPPASSOV, ZHANAT, J. A. C. RAMON and V. PERDEREAU (2015). *Tactile sensing in dexterous robot hands - Review*. Robotics and Autonomous Systems, 74(Part A):195–220.

[KARAMAN et al., 2011] KARAMAN, S., M. WALTER, A. PEREZ, E. FRAZZOLI and S. TELLER (2011). *Anytime Motion Planning using the RRT**. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[KAY, 2016] KAY, JACKIE (2016). *Proposal for Implementation of Real-time Systems in ROS 2*. `http://design.ros2.org/articles/realtime_proposal.html`.

[KERR et al., 2014] KERR, E., T. M. MCGINNITY and S. COLEMAN (2014). *Material classification based on thermal and surface texture properties evaluated against human performance*. In *13th International Conference on Control Automation Robotics Vision (ICARCV)*, pp. 444–449.

[KERZEL et al., 2017] KERZEL, M., M. ALI, H. G. NG and S. WERMTER (2017). *Haptic material classification with a multi-channel neural network*. In *2017 International Joint Conference on Neural Networks (IJCNN)*.

[KIM et al., 2014] KIM, S., A. SHUKLA and A. BILLARD (2014). *Catching Objects in Flight*. IEEE Transactions on Robotics, 30(5).

[KINGMA and BA, 2015] KINGMA, DIEDERIK P. and J. BA (2015). *Adam: A Method for Stochastic Optimization*. In *Proc. ICLR*.

[KOBER et al., 2012] KOBER, J., M. GLISSON and M. MISTRY (2012). *Playing Catch and Juggling with a Humanoid Robot*. In *Proc. IEEE-RAS International Conference on Humanoid Robots*.

[KOC et al., 2018] KOC, OKAN, G. MAEDA, G. MAEDA and J. PETERS (2018). *Online optimal trajectory generation for robot table tennis*. Robotics and Autonomous Systems, 105:121–137.

[KOCH et al., 2015] KOCH, G., R. ZEMEL and R. SALAKHUTDINOV (2015). *Siamese neural networks for one-shot image recognition*. In *Proc. ICML Deep Learning workshop*.

[KOHAVI, 1995] KOHAVI, RON (1995). *A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection*. In *Proc. Int. Joint Conference on Artificial Intelligence*.

[Kreyssig, 2016] Kreyssig, Julia (2016). *Geometrical Calibration of a Tactile Skin on a Humanoid Robot*. Master's thesis, Technische Universität München Fakultät für Informatik.

[Krizhevsky et al., 2012] Krizhevsky, A., I. Sutskever and G. Hinton (2012). *ImageNet classification with deep convolutional neural networks*. In *Proc. NIPS*.

[Labs, 2012] Labs, Sandia National (2012). *The Sandia Hand*. `https://www.sandia.gov/research/robotics/_assets/documents/SandiaHand_Handout_Final.pdf`.

[LaValle, 2006] LaValle, Steven M. (2006). *Planning Algorithms*. Cambridge University Press.

[Leite et al., 2012] Leite, Alexandre Carvalho, B. Schäfer and M. L. de OLiveira e Souza (2012). *Fault-Tolerant Control Strategy for Steering Failures in Wheeled Planetary Rovers*. Journal of Robotics, 2012.

[Lemburg et al., 2011] Lemburg, J., J. de Gea Fernández, M. Eich, D. Mronga, P. Kampmann, A. Vogt, A. Aggarwal, Y. Shi and F. Kirchner (2011). *AILA - design of an autonomous mobile dual-arm robot*. In *2011 IEEE International Conference on Robotics and Automation*, pp. 5147–5153.

[Li and Adelson, 2013] Li, Rui and E. H. Adelson (2013). *Sensing and Recognizing Surface Textures Using a GelSight Sensor*. In *Proc. CVPR*.

[van der Maaten and Hinton, 2008] Maaten, L.J.P. van der and G. Hinton (2008). *Visualizing High-Dimensional Data Using t-SNE*. Journal of Machine Learning Research, 9:2579–2605.

[Maruyama et al., 2016] Maruyama, Yuya, S. Kato and T. Azumi (2016). *Exploring the performance of ROS2*. In *Proc. IEEE Conference on Embedded Software*.

[MathWorks] MathWorks. *The MathWorks*. `http://www.mathworks.com/`.

[Metta et al., 2006] Metta, G., P. Fitzpatrick and L. Natale (2006). *YARP: Yet Another Robot Plattform*. International Journal of Advanced Robotics, 3(1):43–48.

[Metta et al., 2008] Metta, Giorgio, G. Sandini, D. Vernon, L. Natale and F. Nori (2008). *The iCub humanoid robot: an open platform for research in embodied cognition*. In *Proc. of the 8th Workshop on Performance Metrics for Intelligent Systems*.

[Mittendorfer and Cheng, 2011] Mittendorfer, Philipp and G. Cheng (2011). *Humanoid Multimodal Tactile-Sensing Modules*. IEEE Trans. Robot., 27(3):401–410.

[Natale et al., 2016] Natale, Lorenzo, A. Paikan, M. Randazzo and D. E. Domenichelli (2016). *The iCub Software Architecture: Evolution and Lessons Learned*. Frontiers in Robotics and AI, 3:24.

[Newcombe et al., 2011] Newcombe, Richard A., S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges and A. Fitzgibbon (2011). *KinectFusion: Real-time dense surface mapping and tracking*. In *Proc. IEEE International Conference on Mixed and Augmented Reality*.

[Nishiwaki et al., 1997] Nishiwaki, Koichi, A. Konno, K. Nagashima, M. Inaba and H. Inoue (1997). *The Humanoid Saika that Catches a Thrown Ball*. In *Proc. lEEE International Workshop on Robot and Human Communication*, pp. 94–99.

[OptoForce] OptoForce. *OptoForce*. http://optoforce.com/3dsensor/.

[OROCOS] OROCOS. *The Orocos Project*. http://www.orocos.org.

[OSRF, 2015] OSRF (2015). *ROS 2 alpha releases*. https://index.ros.org/doc/ros2/Alpha-Overview/.

[OSRF, 2018a] OSRF (2018a). *ROS 2 Releases*. https://index.ros.org/doc/ros2/Releases/.

[OSRF, 2018b] OSRF (2018b). *ROS 2 Roadmap*. https://index.ros.org/doc/ros2/Roadmap/.

[Ott et al., 2010] Ott, Christian, C. Baumgärtner, J. Mayr, M. Fuchs, R. Burger, D. Lee, O. Eiberger, A. Albu-Schäffer, M. Grebenstein and G. Hirzinger (2010). *Development of a Biped Robot with Torque Controlled Joints*. In *Proceedings of IEEE-RAS International Conference on Humanoid Robots*.

[Ott et al., 2006] Ott, Christian, O. Eiberger, W. Friedl, B. Bäuml, U. Hillenbrand et al. (2006). *A Humanoid Two-Arm System for Dexterous Manipulation*. In *Proc. IEEE/RAS International Conference on Humanoid Robots (HUMANOIDS)*.

[Paikan et al., 2015] Paikan, A., U. Pattacini, D. Domenichelli, M. Randazzo, G. Metta and L. Natale (2015). *A Best-Effort Approach for Run-Time Channel Prioritization in Real-Time Robotic Application*. In *Proc. IEEE International Conference on Intelligent Robots and Systems*.

[PAL Robotics, 2018] PAL Robotics (2018). *TALOS*. http://pal-robotics.com/en/products/talos/.

[Pangercic, 2018] Pangercic, Dejan (2018). *ROS2 for real-time applications (on ROS discourse)*. https://discourse.ros.org/t/ros2-for-real-time-applications/6493/4.

[Pardo-Castellote, 2003] Pardo-Castellote, G. (2003). *OMG Data-Distribution Service: architectural overview*. In *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, pp. 200–206.

[Parusel et al., 2011] Parusel, Sven, S. Haddadin and A. Albu-Schäffer (2011). *Modular state-based behavior control for safe human-robot interaction: A lightweight control architecture for a lightweight robot*. In *Proc. IEEE International Conference on Robotics and Automation*.

[Quigley et al., 2009] Quigley, Morgan, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler and A. Ng (2009). *ROS: an open-source Robot Operating System*. In *Proceedings of the Open-Source Software workshop at the International Conference on Robotics and Automation (ICRA)*.

[Racket] Racket. *Racket*. http://racket-lang.org.

[Ratliff et al., 2009] Ratliff, Nathan, M. Zucker, J. A. D. Bagnell and S. Srinivasa (2009). *CHOMP: Gradient Optimization Techniques for Efficient Motion Planning*. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*.

[Ravi and Larochelle, 2017] Ravi, Sachin and H. Larochelle (2017). *OPTIMIZATION AS A MODEL FOR FEW-SHOT LEARNING*. In *Proc. International Conference on Learning Representations*.

[Reppy, 1999] Reppy, J. H. (1999). *Concurrent Programming in ML*. Cambridge University Press.

[RILEY and ATKESON, 2002] RILEY, MARCIA and C. G. ATKESON (2002). *Robot Catching: Towards Engaging Human-Humanoid Interaction*. Autonomous Robots, 12:119—128.

[ROCK] ROCK. *Rock: the Robot Construction Kit*. `https://www.rock-robotics.org`.

[ROS2] ROS2. *ROS 2*. `https://index.ros.org/doc/ros2/`.

[RTI] RTI. *RTI Connext*. `https://www.rti.com/products`.

[SAKAGAMI et al., 2002] SAKAGAMI, Y., R. WATANABE, C. AOYAMA, S. MATSUNAGA, N. HIGAKI and K. FUJIMURA (2002). *The intelligent asimo: system overview and integration*. In *Proc. IEEE International Conference on Intelligent Robots and Systems*.

[SALEHIAN et al., 2016] SALEHIAN, SEYED SINA MIRRAZAVI, M. KHORAMSHAHI and A. BILLARD (2016). *A Dynamical System Approach for Catching Softly a Flying Object: Theory and Experiment*. IEEE Transactions on Robotics.

[SCHMITZ et al., 2010] SCHMITZ, ALEXANDER, M. MAGGIALI, L. NATALE, B. BONINO and G. METTA (2010). *A Tactile Sensor for the Fingertips of the Humanoid Robot iCub*. In *Proc. IEEE International Conference on Intelligent Robots and Systems*.

[SHADOW] SHADOW. *Shadow Hand*. `https://www.shadowrobot.com`.

[SINAPOV et al., 2011] SINAPOV, JIVKO, V. SUKHOY, R. SAHAI and A. STOYTCHEV (2011). *Vibrotactile Recognition and Categorization of Surfaces by a Humanoid Robot*. IEEE Transactions on Robotics, 27(3):488–497.

[SMITH and CHRISTENSEN, 2007] SMITH, CHRISTIAN and H. I. CHRISTENSEN (2007). *Using COTS to Construct a High Performance Robot Arm*. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[SNELL et al., 2017] SNELL, JAKE, K. SWERSKY and R. S. ZEMEL (2017). *Prototypical Networks for Few-shot Learning*. CoRR, abs/1703.05175.

[SPELLUCCI, 1998] SPELLUCCI, P. (1998). *A Sqp Method For General Nonlinear Programs Using Only Equality Constrained Subproblems*. MATHEMATICAL PROGRAMMING, 82:413–448.

[SPELLUCCI, 1999] SPELLUCCI, P. (1999). *DONLP2 short users guide*. ftp://ftp.mathematik.tu-darmstadt.de/pub/department/software/opti/DONLP2.

[STRESE et al., 2016] STRESE, MATTI, C. SCHUWERK, A. IEPURE and E. STEINBACH (2016). *Multimodal Feature-based Surface Material Classification*. IEEE Transactions on Haptics, PP(99).

[SYNTOUCH] SYNTOUCH. *SynTouch*. `www.syntouchllc.com`.

[TEKSCAN] TEKSCAN. *Tekscan*. `www.tekscan.com`.

[TOBERGTE et al., 2009] TOBERGTE, ANDREAS, R. KONIETSCHKE and G. HIRZINGER (2009). *Planning and Control of a Teleoperation System for Research in Minimally Invasive Robotic Surgery*. In *IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*.

[TOBERGTE et al., 2010] TOBERGTE, ANDREAS, G. PASSIG, B. KUEBLER, U. SEIBOLD, U. A. HAGN, F. A. FRÖHLICH, R. KONIETSCHKE, S. JÖRG, M. NICKL, S. THIELMANN, R. HASLINGER, M. GROEGER, A. NOTHHELFER, L. LE-TIEN, R. GRUBER, A. ALBU-SCHÄFFER and G. HIRZINGER (2010). *MiroSurge—Advanced User Interaction Modalities in Minimally Invasive Robotic Surgery*. Presence: Teleoperators and Virtual Environments, 19(5):400–414.

[TULBURE and BÄUML, 2018] TULBURE, ANDREEA and B. BÄUML (2018). *The TactMat Dataset: DLR's Robotic Tactile Material Classification Dataset*. `dlr-alr.github.io/dlr-tactmat`.

[VAHRENKAMP et al., 2015] VAHRENKAMP, NIKOLAUS, M. WÄCHTER, M. KRÖHNERT, K. WELKE and T. ASFOUR (2015). *The robot software framework ArmarX*. it - Information Technology, 57(2).

[VINYALS et al., 2016] VINYALS, ORIOL, C. BLUNDELL, T. LILLICRAP, K. KAVUKCUOGLU and D. WIERSTRA (2016). *Matching Networks for One Shot Learning*. In LEE, D. D., M. SUGIYAMA, U. V. LUXBURG, I. GUYON and R. GARNETT, eds.: *Advances in Neural Information Processing Systems 29*, pp. 3630–3638. Curran Associates, Inc.

[VOGEL et al., 2015] VOGEL, J., S. HADDADIN, B. JAROSIEWICZ, J. SIMERAL, D. BACHER, L. HOCHBERG, J. DONOGHUE and P. VAN DER SMAGT (2015). *An assistive decision-and-control architecture for force-sensitive hand–arm systems driven by human–machine interfaces*. International Journal of Robotics Research, 34(6):763–780.

[VOGEL et al., 2011] VOGEL, JÖRN, C. CASTELLINI and P. VAN DER SMAGT (2011). *EMG-Based Teleoperation and Manipulation with the DLR LWR-III*. In *Proc. IEEE International Conference on Intelligent Robots and Systems*.

[VOGEL et al., 2010] VOGEL, JÖRN, S. HADDADIN, J. D. SIMERAL, S. STAVISKY, D. BACHER, L. R. HOCHBERG, J. P. DONOGHUE and P. VAN DER SMAGT (2010). *Continuous Control of the DLR Light-weight Robot III by a human with tetraplegia using the BrainGate2 Neural Interface System*. In *International Symposium on Experimental Robotics*.

[WAELVELDE et al., 2003] WAELVELDE, HILDE, W. DE WEERDT, P. DE COCK and B. SMITS-ENGELSMAN (2003). *Ball catching. Can it be measured?*. Physiotherapy Theory and Practice, 19:259–267.

[WETTELS et al., 2008] WETTELS, N., V. SANTOS, R. JOHANSSON and G. LOEB (2008). *Biomimetic tactile sensor array*. Advanced Robotics, 22(8):829–849.

[WILLOW GARAGE] WILLOW GARAGE. *PR2 - Personal Robot 2*. `www.willowgarage.com`.

[WIMBÖCK et al., 2009] WIMBÖCK, THOMAS, D. NENCHEV, A. ALBU-SCHÄFFER and G. HIRZINGER (2009). *Experimental Study on Dynamic Reactionless Motions with DLR's Humanoid Robot Justin*. In *Proc. IEEE International Conference on Intelligent Robots and Systems*.

[XU et al., 2013] XU, D., G. LOEB and J. FISHEL (2013). *Tactile identification of objects using Bayesian exploration*. In *Proc. IEEE International Conference on Robotics and Automation*.

[XU et al., 2014] XU, YAN, T. MO, Q. FENG, P. ZHONG, M. LAI and E. I. C. CHANG (2014). *Deep learning of feature representation with multiple instance learning for medical image analysis*. In *Proc. IEEE Int. Conf on Acoustics, Speech and Signal Processing*.

Appendix A

List of Videos

We provide a list of videos of the presented work and their weblinks for easy access.

- ”Robust Material Classification with a Tactile Skin Using Deep Learning”, 2016, additional video for [4].
  `https://www.youtube.com/watch?v=623yRPx9Pkc`
- ”Agile Justin: An Upgraded Member of DLR’s Family of Lightweight and Torque Controlled Humanoids”, 2014, video of contribution [5].
  `https://www.youtube.com/watch?v=Fl2N6yZrk1o`
- ”Agile and Rollin’ Justin Playing Ball”, 2012, [BÄUML, 2012].
  `https://www.youtube.com/watch?v=93WHRSKg3gE`
- ”Catching Flying Balls and Preparing Coffee: Humanoid Rollin’Justin Performs Dynamic and Sensitive Tasks”, 2011, video of contribution [7].
  `https://www.youtube.com/watch?v=R6pPwP3s7s4`
- ”Kinematically Optimal Catching a Flying Ball with a Hand-Arm-System”, 2010, accompanying video of [8].
  `https://youtu.be/ssR7rIKajeo`