Technische Universität München
Fakultät für Informatik
Lehrstuhl für Echtzeitsysteme und Robotik

Architecture and Methodology for

Sensor-Based Robotics on Lie Algebras

Thomas Müller

Technische Universität München
Fakultät für Informatik
Lehrstuhl für Echtzeitsysteme und Robotik

Architecture and Methodology for
Sensor-Based Robotics on Lie Algebras

Thomas Müller

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Disseration.

Vorsitzender:	Univ.-Prof. Dr. Thomas Neumann
Prüfer der Dissertation:	
1.	Univ.-Prof. Dr. Alois Knoll
2.	Univ.-Prof. Dr. Rüdiger Dillmann, Karlsruher Institut für Technologie

Die Dissertation wurde am 18.06.2012 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 11.02.2013 angenommen.

Zusammenfassung

Die Arbeit beschäftigt sich mit kontinuierlichen Transformationsgruppen und deren Algebren, den Lie Algebren, in der sensorbasierten Robotik. Dabei zeigt sich in einer Referenzalgebra von Sensorik und Aktorik, dass mit Hilfe eines neuartigen Deskriptors für Aktivierung Mechanismen selektiver Aufmerksamkeit für autonome Robotersysteme modelliert und sogar reflexartige sensormotorische Assoziation abgebildet werden können. Außerdem erlaubt das Konzept die Integration von Entscheidungen höherer koordinativer Instanzen in die sensormotorische Verarbeitungskette.

Vorgestellte theoretische Methoden, abgeleitete Algorithmen und Anwendungen werden in zwei Szenarios der Robotik implementiert und diskutiert. Im ersten Szenario wird prototypisch ein sensorbasiertes System für den Bereich industrielle Automatisierung entwickelt, während im zweiten Szenario Anwendungen im Bereich Mensch-Maschine Interaktion gezeigt werden.

Abstract

The thesis discusses continuous transformation groups and their algebras, the Lie algebras, for sensor-based robotics. Here it is shown in a reference algebra for sensor and actuator subsystems, that a novel descriptor for activation enables modeling of selective attention mechanisms for autonomous robotic systems, and eventually reflexive sensorimotor association. Furthermore the concept allows for integration of decision-making instances into the reflexive association tool-chain.

Presented theoretical methods, derived algorithms and applications are implemented and discussed in two robotic scenarios. In the first a prototypical sensor-based system for industrial automation is developed, while in the second applications in the field of human-robot interaction are shown.

Acknowledgements

First of all, I want to thank my supervisor, Professor Alois Knoll, for providing me the opportunity to develop my own ideas and prepare this thesis. I am very thankful to Professor Dillmann for accepting to be my external reviewer and for interesting discussions in the defence.

Many thanks go to the entire group Embedded Systems and Robotics at the TU München for the pleasant atmosphere and valuable technical arguments.

I also thank family, my parents Christine and Peter, my brother Uwe, and my grandma Hilde, for supporting me on becoming a "G'studierta".

Finally, very special thanks go to my girlfriend Katharina for the extra strength and love necessary to get this thesis done.

Thomas Müller
München
12.07.2013

This thesis is dedicated to Katharina, my beloved girlfriend and expectant mother of our child.

Where would I be today without you?

Contents

Chapter 1

Introduction

Contents

ROBOTS are common in todays industrial production setups. They accompany assembly lines all over the world, as they have interesting properties for production processes: they never tire, provide high accuracy, and are able to work in environments not suitable for humans. Still, today's robots are often limited to very specific tasks, as programming them accurately often is a non-trivial task and thus only relatively simple, repetitive tasks are conducted autonomously by robots.

The main reason for this is the lack of capability for intelligent reaction on environmental inputs. The cause is simple: the classical robot has neither means to perceive its environment nor sufficient "intelligent" routines to react on unforeseen events or dynamically adjust its behavior with respect to processed input information. Alongside, many issues arise in typical robotic setups. Humans are forbidden access to the workspace of robots for security reasons; reconfiguration is expensive and time-consuming; dynamic environments are difficult to handle; and flexible production processes cannot be implemented easily, to name but a few.

Therefore, integrating sensory devices for specific tasks gets increasingly popular in robotics, but computaional cost of extensive input data processing and realtime reaction requirements prohibit general usage. Nevertheless, the direction goes from "dumb" industrial machinery[1] towards social humanoid robot companions. Thus this traditionally also requires increasing complexity in the algorithms applied to control these robots. To handle this complexity recent approaches following the *embodiment* paradigm argue, that morphological agent[2] constraints can reduce the computational costs for controlling and increase flexibility of an agent. This is facil-

[1] In fact these systems are often highly sophisticated considering their level of programmed expert task knowledge, but rather dumb with respect to their cognitive ability.

[2] The term "agent" is used according to embodiment terminology, i.e., whenever the distinction between a technical and biological system shall not be explicitly made.

itated with the tight integration of a body with its controller. The body eventually masters complex behavior by exploiting its morphology through reflexive strategies. Unfortunately, real-world mechanical and industrial robotic setups usually do not provide for the necessary properties of embodied agents, such as underactuated muscle-tendon systems with physical feedback mechanisms.

To overcome this from a theoretical, computational perspective, the problem to be solved is in general inconsistent representation of actuator control and sensory input data. Fusion and sensorimotor coordination, which are the basis of embodied intelligence, however mostly require data association on a low level on input and output channels simultaneously. This thesis therefore introduces a consistent and elegant mathematical approach for modeling and managing robot manipulation and sensory perception capabilities based on group theory and Lie algebras.

1.1 Contribution

The theoretical approach proposed in this thesis includes original work on applying continuous transformation groups and their tangential vector spaces at the identity group element, the Lie algebras, to sensor-based robotic systems. For such applications a descriptor structure based on Lie algebras is derived. Means for generating, combining, and evaluating these descriptors are discussed on a mathematical level.

Then, this work proposes applications of the Lie algebraic activation features to different domains of sensor-based robotics. This includes modules for efficient realtime manipulator control, sensor data (pre-)processing, and sensorimotor association and coordination (see Figure 1.1).

- For the manipulator control domain, a realtime path generation system for freely moving Cartesian targets is developed. It allows for receiving target pose updates from activation features at any time, while still generating smooth end-effector paths according to the hardware specifications and motion constraints.
- Thereafter applying the activation features to early processing in the perception domain is presented. Here, using the Lie algebraic constructs an attention mechanism is implemented, which includes bottom-up excitation and top-down feedback, and is similar to the human attention system. It is shown, how this attention based vision system can be used to dramatically improve performance on workspace surveillance on the one hand and multiple interaction partner recognition tasks on the other.
- Finally, in the coordination domain the activation features can be used to real-

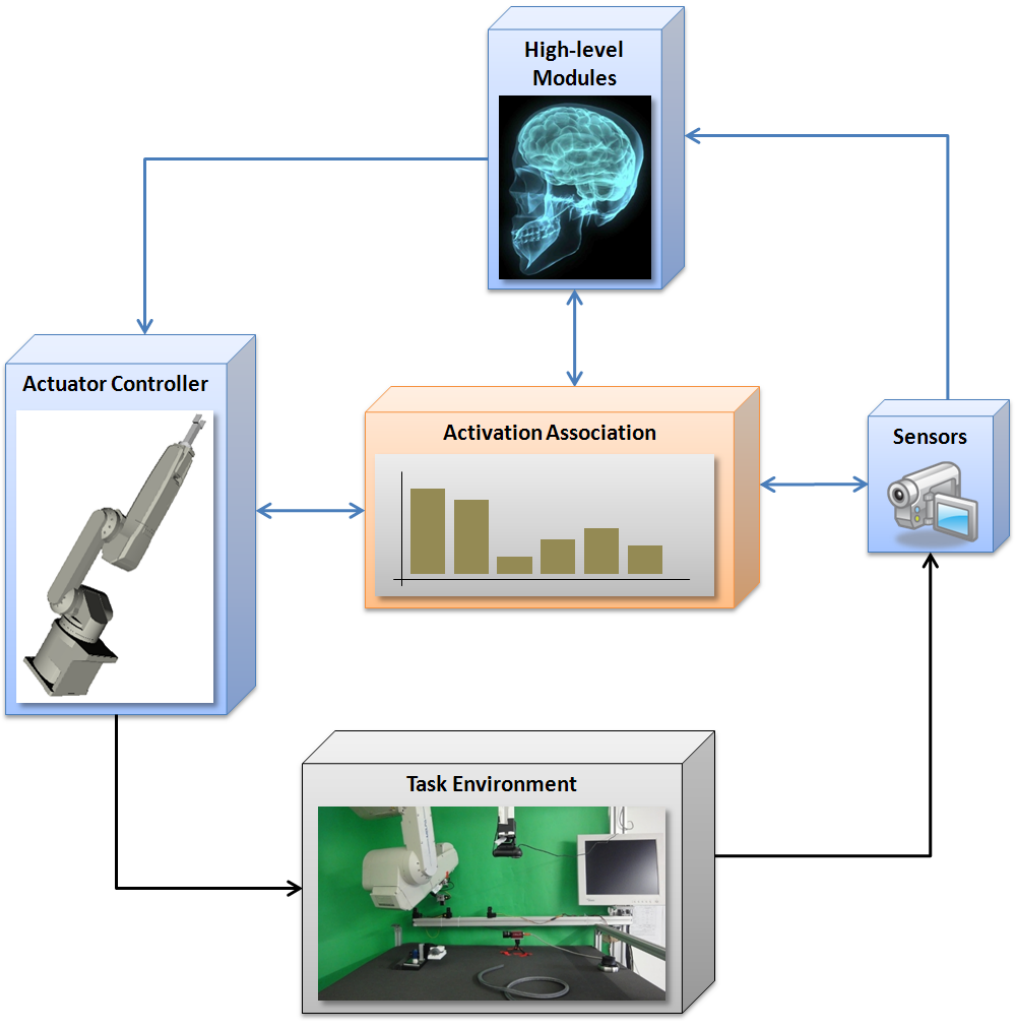

Figure 1.1 *Gobal theoretical architecture concept of a sensorimotor coordinated system.*

ize direct sensorimotor association as necessary for traditional visual servoing applications. In addition to that, the activation features allow for seamless integration of intentional, and therefore high-level cognitive inputs to the reflexive servoing system on a sensorimotor association level. Unifying the representation of actuator-, sensor-, and cognitive activation with the Lie algebraic features, and multi-directional combination and propagation by an association layer facilitate this capability.

1.2 Scenarios and Demonstrations

The proposed theoretical methods and derived algorithms are implemented in two real world robotic scenarios. One scenario includes transferring research findings into an industrial automation prototype, while the other scenario is dedicated to research on human-robot interaction. Both application scenarios originate from and correspond to externally funded research projects.

The first scenario is within the SFB 453-T4 project. The transferproject T4 on "Automated, Robot-Assisted Handling of Limp and Deformable Objects" within the collaborative research center SFB 453 on "High-Fidelity Telepresence and Teleaction" is supported by the German Research Foundation (DFG). Aiming towards industrial automation, in fact many aspects of the proposed work are implemented in different applications within this scenario. For example, the robotics framework for distributed processing, algorithms for detection and tracking of linear structures, task coupling and abstraction, fault and error recovery, as well as realtime path generation on freely moving targets, or vision based servoing methods are applied here.

The second scenario is within the JAST project. The *Framework Program 6* project (FP6-003747-IP) on "Joint-Action Science and Technology" is supported by the European Union. In this scenario a computer vision and dialog based human-robot interaction system is built. For example parallelization on multi-core machines are implemented in a workspace observation subsystem; or early information processing, activation correspondence and attention condensation mechanisms with high-level inhibitory feedback are integrated into a subsystem for interaction partner recognition.

The presented applications in each of the scenarios of sensor-based robotics show the applicability of proposed theoretical and technical claims and in deed reveal significant advances to the state of the art in several fields of research. The scenarios also demonstrate successful integration of the discussed novel concepts with existing state-of-the-art techniques and systems.

1.3 Structure of the Thesis

The thesis essentially comprises the following four parts.

- Chapter 2
  Background and in-depth information on group theory in general and the necessary details on continuous transformation groups and in particular their algebras, the Lie algebras, are provided in Chapter 2. Alongside mathematical concepts based on Lie groups and algebras with respect to robotic applications are discussed.

- Chapter 3
  This chapter introduces methodology for deriving Lie algebraic features from activation in the sensor, actuator, and coordination domain. It furthermore shows how these domains can be unified in an association layer, i.e., how for example sensory stimuli can be translated into motor output commands and vice versa. The mathematics necessary for this process are also introduced here.

- Chapter 4
  Applications and algorithmic methods based on differential algebras and Lie groups for the different domains of sensorimotor robotics are presented in Chapter 4. Here, useful applications, in particular realtime path generation for the manipulator domain, attention mechanisms for the perception domain, and sensorimotor association strategies for reflexive and coordinated behavior are derived.

- Chapter 5
  The final chapter gives details on the two research scenarios, SFB and JAST, mentioned above and discusses results with respect to experimental applications. Then it summarizes the contribution and draws a conclusion alongside showing possible directions for future research in the field.

The appendix finally shows additional details on transformation group representations for robotics in Appendix A, and gives a short introduction to the software framework implemented as part of this thesis in Appendix B.

Chapter 2

Background and Related Work

Contents

In Section 2.1 this chapter starts with a coarse introduction to robot perception and action, as well as to relevant influential theories of (artificial) intelligence based on embodiment and sensorimotor coordination. An introduction to the mathematic fundamentals of group theory in general and continuous transformation groups and their algebras in particular follows in Section 2.2. Readers familiar with the concepts of group theory, embodied cognition theory and models for cognitive systems may very well continue with Chapter 3 directly and scroll back if necessary.

2.1 Sensor-Based and Autonomous Systems

As [Noë 2004] claim, perception is essentially processing an input data stream originating from some representation of transformation, as sensors in general receive stimuli from events caused by transformation of the agent environment, while action means actively producing a transformation of the environment respectively. On the lowest level sensorimotor association and coordination, and eventually development of cognitive abilities on higher levels, may hence in principle be based on processing transformation. Related models, supporting this claim or explaining development of cognitive abilities, are discussed in the following.

Current neuroscience research aims at modeling neural processes and give explanations for development of intelligence or at least intelligent behavior. Most theories agree, that action, perception, and the coupling of both for sensorimotor coordination is an important, if not the most important prerequisite for this task in biological systems. Therefore, because frequently biological agents outperform artificial ones, biologically inspired models also seem plausible for development of autonomous robotic systems.

Some influential models for this thesis are thus briefly reviewed in this section. How-

ever it has to be emphasized, that this work does not claim to give a new theory of cognition nor a model of sensorimotor coordination. It is rather to provide an elegant theoretical, mathematical method which implements sensorimotor association in a real-world robotic system, as in general such a system does not provide the necessary hardware or low-level architecture by default.

The next sections also show relevant related research in the field of robot action, i.e. motion generation and action abstraction, and perception, where in particular active approaches and attention mechanisms are reviewed.

2.1.1 Autonomous Systems and Intelligence

In order for a system to act autonomously in its task environment, to some degree, the system has to show intelligent behavior. First, this means, the system needs to be able to perceive relevant data from its environment; second, it needs to adapt its behavior to unforeseen events, i.e., it has to realize that such events occurred and plan its action accordingly. Towards such autonomy different theoretical models have evolved over the recent decades. The most influential state-of-the-art ones for this work are elaborated in the following.

Embodiment

Embodiment theory in essence postulates that an autonomous agent can only be built and moreover, a system can only develop intelligence, if and only if it has means to actively interact with its environment. Furthermore, the system needs to have sensory devices to perceive its interaction [Brooks 1991, Clark 1997, Anderson 2003]. According to [Pfeifer and Bongard 2005, pp. 100], the following basic design principles then apply for embodied agents.

The *three-constituents* design principle claims, that one has to consider three essential constituents for designing an intelligent embodied agent.

1. The ecological niche for the agent has to be defined. The ecological niche comprises the agents environment, e.g., a factory, a museum, or a supermarket.
2. The desired behavior has to be defined. This states the set of behaviors and tasks that the system has to perform. Together with its ecological niche this forms the task environment.
3. The pyhsical design of the agent. The system morphology has to be defined in accordance to the requirements of its task environment.

The *complete-agent principle* of embodiment theory claims, that it is often clumsy

and counter-productive to decompose the problem of developing intelligent behavior into its pieces, namely action, perception and cognition. Instead, it is proposed, that an agent needs to interact autonomously in its task environment (explore it) and must simultaneously perceive its interaction through sensory devices [Pfeifer and Iida 2004].

The *cheap design* principle states that it will be much more cost effective, if an agent takes advantage of its morphology and its ecological niche. The system then saves a lot of computing power (cognitive resources) by just exploiting its design and the properties of its physical environment [Pfeifer and Gómez 2005].

The *redundancy* principle in embodiment on the one hand requires an agent to "function on the basis of different physical processes" [Pfeifer and Bongard 2005, pp. 113]. This means sensor devices have to be designed that perceive the dynamics of the agent environment based on different properties, e.g., vision (electro-magnetic waves) and haptics (tactile sensation). On the other hand it states that there has to be at least a partial overlap on the functionality from those sensors – the same task could be achieved by using different modalities.

The *sensorimotor coordination* principle essentially states, that structure in the sensory stimulation is generated by an agent performing interactions with its environment. An agent moving in its environment generates specific structure in the data stream of its sensory devices. This is also valid vice-versa, where sensory stimulation induces structured interaction in order to master tasks in an environment. A prominent example for this is walking on uneven ground. In order to master this task, sensing of the (humanoid) agent's feet touching the ground enables the agent to react on the stimuli and walk properly. Sensorimotor coordinated behavior thus always involves the simultaneous processing and association of input (perception) and output (action).

A psychological perspective of this principle is discussed in [O'Regan and Noë 2001], where it is proposed that any perception through a sensory device, particularly in the visual apparatus in biological system is always correlated to actively causing changes in the stimuli. It is claimed that only by moving our eyes, i.e. adjusting our field of view and foveating on objects of interest ("active inquiry and exploration" [Noë 2004]), we are able to actually see objects. This enactive approach is discussed with respect to object recognition, color perception and in general experiencing and understanding the environmental structure.

Embodiment's design principle of *ecological balance* [Hara and Pfeifer 2000] refers to a match of complexity within an agent's design (actuators and sensors) and the

task environment. Given a simple task, it is not necessary to apply sophisticated manipulators and sensors – this would rather hinder efficient behavior. On the other hand, the principle states that also there shall be a balance between actuator and sensor materials and morphology, control and environment [Pfeifer 2000]. If an agent has complex actuators it is for example also necessary to apply sensors that may perceive the interaction with the environment caused by these actuators, as well as the cognitive architecture to process this information [Ishiguro and Kawakatsu 2003].

Traditionally, hierarchical robot control architectures implement a controller program, which calls subroutines for sensation (e.g. retrieval of camera images), then performs processing on this data, plans actions and finally calls a motor subroutine for actually moving the actuators. This approach is not only hierarchical, but also sequential.

The principle of *parallel, loosely coupled processes* states, that intelligence may only emerge from systems that break with this traditional approach (see Figure 2.1[1]). It is claimed, that an embodied intelligent system instead has to implement a *subsumption architecture* (see Figure 2.2). This means, all processes run in parallel, even if their output is not currently required on higher levels. and the coupling of low-level processes via higher-level processes is only loose: coordination of (low-level) processes is not achieved by complex internal (higher-level) processing, but emerges from the interaction with the environment.

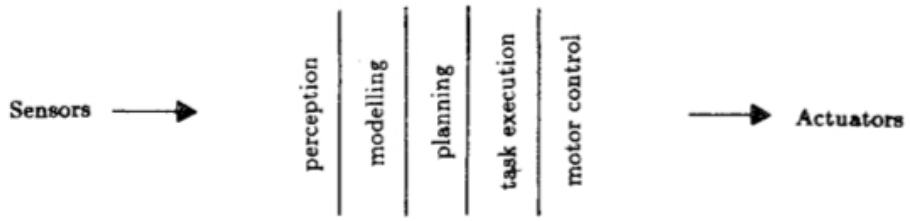

Figure 2.1 *Traditionally, functional modules for robot control are decomposed with respect to a sequential processing pipeline.*

Finally, the *value* design principle states an agent needs some kind of a value system for autonomous learning and self organization [Pfeifer and Bongard 2005, pp. 137].

[1] Figure 2.1 and Figure 2.2 are reprints from [Brooks 1986].

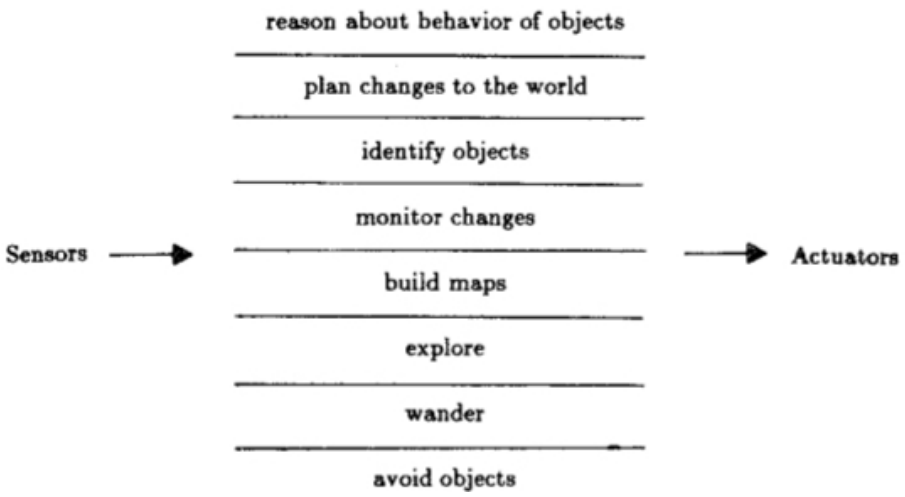

Figure 2.2 *Subsumption architecture proposes a decomposition based on tasks and behaviors.*

Memory-Prediction and Generative Models

The last section introduced the embodiment model which tries to explain the development of intelligence in a natural or an artificial agent. The model reviewed below on the other side tries to explain the algorithms or processes implementing intelligence instead. Although many generative neural models of intelligence and coupling of perception and action have been developed and, at least partially, implemented (see e.g. [Hinton 2007, Dayan and Abbott 2001] for an overview), the most influential for this thesis is the memory-prediction model from [Hawkins 2004].

The memory-prediction model gives an alternative theory of intelligence (or the emergence of intelligence), which nevertheless partially overlaps with the embodiment theory. However, it is more a model of neural processes in the neocortex, rather than a theory of creating intelligent behavior for artificial systems. Yet, some implications of the memory-prediction model seem relevant to this thesis and several aspects inspire the technical solutions presented later.

The first claim of the memory-prediction model is, that intelligence is not a matter of computational power. It was found that the operating time scale of biological neurons is much slower than that of electronic circuits (ca. 10^{-2} vs. 10^{-7} seconds). Therefore speed and computational power cannot account for intelligence. It is rather massively parallel processing and the layered prediction of memorized structures that makes the biological system efficient.

Findings from neuroanatomy furthermore show, that the physical structure of the neocortex is columnar [Mountcastle 1978] and in fact relatively uniform all over.

Summary 2.1 – **Embodiment**

The most relevant implications of embodiment for this thesis are the following.

- Tight coupling of body (morphology), perception and cognition is essential for development of intelligence. A complete agent has to consider action/perception simultaneously – no decomposition is necessary, in fact it is obstructive.
- "Automatically" finding correlation between action/perception stimuli is essential (sensorimotor coordination).
- Parallel, loosely coupled processes are inherent: in the subsumption architecture everything runs in parallel, and only relevant information is propagated, while any other is inhibited.

The memory-prediction model thus claims there is a *single, general algorithm* operating everywhere in the cortex to enable intelligence (perception, action, planning and learning). This idea is supported by the sensory substitution experiments [Bach-y-Rita 1967], where a haptic display device was developed and placed on a blind subjects tongue to substitute for the visual input (the subjects actually "saw" objects by feeling pressure on their tongue).

The general algorithm according to the memory-prediction model is based on auto-associativity and Hebbian learning [Hebb 1949]. Connections and associations are strengthened, whenever they occur simultaneously. Then, abstraction ("naming") correlates to building *invariant internal representations* of associated inputs patterns. These representations are formed hierarchically during learning. Bottom-up in the hierarchy, spatial specifity gets lost in favor of an invariant representation, short time coverage incrementally increases towards larger time coverage, and details and sensory features become objects (see Figure 2.3[2]).

So the memory first creates a model of the world, i.e. an invariant representation. Once then sensory input is related to a stored model, it is possible to make continuous predictions. As long as these predictions (expectations) match the likewise continuous sensory stimuli, the system is in a consistent state and uses no further

[2] Figure 2.3 is taken from [Hawkins 2004, p. 77].

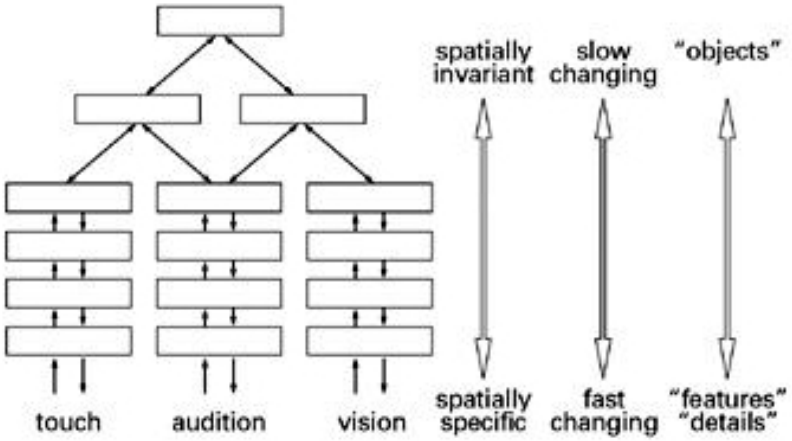

Figure 2.3 *Different spatial/time scale and abstraction reference frames.*

computational resources. Only if an unexpected stimulus is received, which is not consistent with the current predicted stimulus, this is propagated upwards in the hierarchy. Whenever the next layer is able to incorporate this change into its model, predictions are adapted and the system is in a consistent state again. If this process fails, the inconsistency is propagated upwards again. Eventually, at the top-level of the hierarchy it is then decided, whether the inconsistency corresponds to a new model or the model has to be adjusted globally. The general idea is to delay a decision on an inconsistency and first wait for higher-level feedback. This correlates strongly to the subsumption architecture presented earlier, as here as well it is postulated, that consistent activation is suppressed from higher-levels. This effect has also been investigated on in biological systems [Posner and Cohen 1984, Tipper and Kingstone 2005], where it is commonly called *inhibition of return*.

Before learning, a column in the neocortex can only become active if driven by a bottom-up stimulus (a layer four cell) [Hawkins 2004, pp. 101]. After learning, the column can become partially active via memory. When a column becomes active via top-down feedback, it is anticipating being driven from below. With respect to generating action in and interaction with the environment, the memory-prediction model claims the following: action / behavior is a "highly coordinated firing sequence along layer five cells within the motor cortex" [Hawkins 2004, pp. 98]. This causes for example muscle contraction. So actuation is not actually planning a command sequence, but automatically happens by memorizing and predicting a pattern and subsequently creating sequences of activations in the motor cortex which then (also in anticipation of further activation) fires the motor commands.

A final claim of the memory-prediction model is that intelligence is not dependent on

the physical material of implementation of the algorithm. Any machine or biological system providing the architecture to perform the processing steps of the algorithm eventually develops intelligence. Thus, intelligent machines may evolve in shapes, appearances, and with interaction skills likely to be very different from the ones of humans.

Summary 2.2 – **Memory-Prediction Model**

The most relevant aspects of the memory prediction model for this thesis are

- a uniform (columnar) neural architecture implies a common, general algorithm for development of intelligence,
- building an invariant internal representations of associated inputs patterns ("naming" things) is the learning part,
- the prediction part is "anticipation" of activation by integration of patterns with delayed feedback (of recent activation),
- only unexpected, unpredicted input is propagated upwards for learning, while expected activation is suppressed ("inhibition of return"), and
- anticipated activation can introduce motion by mutual excitation.

2.1.2 Robot Action

Towards controlling a robot for interaction within its environment, several aspects have to be considered. First, the low-level control of the hardware is essential, which is mainly a technical problem, i.e., defining controller interfaces, sending motor commands, and the like. Next, the robot motion has to be coordinated somehow. This usually involves modeling the kinematic structure, computing a trajectory in space, and applying the control commands according to inverse kinematics. An incremental approach may then abstract from primitive motion and trajectory planning and lead towards complex behavior specifications.

Motion Generation and Planning

In the field of robot motion generation and (path) planning a vast amount of research from the last decades is present in literature. An extensive survey of the field is for example given in [Latombe 1991].

In general, one can distinguish between two classes of systems: holonomic and non-holonomic systems [Laumond 1998]. First, in non-holonomic systems the actuated degrees of freedom are not independent. For example, a car only has two means of actuation, steering and linear motion. Yet, these are not independent, as the car may only rotate around its axis while also changing the position. Second, holonomic systems are fully actuated in their configuration space, i.e., given a zero current velocity, a robotic system can actually move to any pose in its space of reference directly (the degrees of freedom of the actuator match those of the environment). Some also attribute underactuated systems to be a separate class [Choset et al. 2005], because their actuators are designed with additional degrees of freedom with respect to the environment, these systems are also called redundant. In a redundant system not only one configuration exists to match a desired final destination, but the number is infinite.

This thesis does not need to explore this field of research in great depth. Instead, a methodology is derived for efficient path generation in realtime. Generally, we determine offline and online approaches. Offline methods are especially useful for repetitive tasks, as the path can be optimized beforehand and then simply be executed as often as required. Still, finding a suitable path with these planners may be a non-trivial task. Common approaches use exploration or exploitation strategies, or a combination of both (see [Rickert et al. 2008] for an extensive discussion). The focus of interest with online, i.e. realtime, approaches is slightly different. Here, one tries to approximate an optimal solution while still offering the flexibility required for highly dynamic environments. Within these environments new targets and / or obstacles may be introduced at any time during a movement and the robot is required to act in realtime upon these new constraints. Movement planning thus has to consider the current operational and configuration space pose and velocities and combine these with the target and obstacle information. Common online approaches use artificial potential fields [Khatib 1985, Khatib 1987] or fluid dynamics [Mayer et al. 2007].

For robot control, i.e. actuating the robot joints in order to reach a previously computed end-effector pose with a corresponding joint configuration, we determine the problems of computing forward and inverse kinematics. While forward kinematics

are relatively easy and efficiently to compute, e.g., using the standard *Denavit-Hartenberg* (DH) convention [Denavit and Hartenberg 1955], the problem of inverse kinematics is a lot more complicated. Here, *all* approaches, without exception, face the problem of singularities when having as much as or more than actuator space degrees of freedom. Still there are several acceptable solutions "on the market" that suffice every day tasks. Amongst these solutions, in principal three approaches can be determined. Algebraic and geometric methods try to find a solution to the inverse kinematics problem by successive inversion of the DH-transforms. Algebraic methods produce trigonometric non-linear equations to be solved and geometric methods reduce the large problem to a series of plane geometry problems. These closed-form approaches are able to find all solutions to the problem, but they may be inefficient to compute and difficult to evaluate, as a suitable solution has to be found from the (probably infinite) set of solutions provided. The third category, numerical approaches, provide a compromise being both efficient to compute and accurately enough to be useful for real-world scenarios. Here, an approximate solution is derived from an underlying non-linear optimization problem. A magnitude of algorithms for such optimization problems have been proposed, including for example *Downhill-Simplex* [Nelder and Mead 1965], *Levenberg-Marquardt* [Levenberg 1944], and many more in [Press et al. 2002].

The group theory based approach presented here derives a relatively easy solution for inverse kinematics in a scenario, where the degrees of freedom of the actuator match those of the environment. This is significant, as it is a closed-form approach which is still efficient to compute. In combination with a compositional realtime path planner, the system may in an unconstrained environment, given some current state of the system, always produce an infinitesimal increment yielding smooth reaching towards a possibly dynamic target. In practice, of course, the infinitesimal increments are discretized with respect to the update interval of the robot hardware and thus the approach is, strictly speaking, still an approximation. Nevertheless, motion planning is elegant, as the approach only computes in the tangent space of the transformation manifold.

Action Abstraction and Complex Behavior

Regarding incremental automation and behavior abstraction, Arkin's introduction to the principles, design, and practice of intelligent behavior-based autonomous robotic systems [Arkin 1998] is one of the first surveys of this robotics field (others follow, e.g., [Graves and Czarnecki 2000]). Tools and techniques central to the development of this class of systems are presented there. His work covers the use of

knowledge and learning in autonomous robots, behavior-based robot architectures, modular perception, and future trends in robot intelligence. The central idea is, that complex behavior can be composed from primitive prototypes (or action primitives) by combining characteristic, distinct sequences of primitives. If for example reaching for an object and closing the manipulator hand afterwards always occur in sequence, a "grasp" behavior can be abstracted from that. This abstract behavior can in turn then eventually be utilized as a primitive for even complexer behavior, instead of coordinating the sequence of primitives over and over again. A previous attempt to implement such a system is e.g. discussed in [Radi et al. 2010].

Of course, many others have also contributed significantly to this field of research, e.g., [Laaksonen et al. 2010] propose an abstraction architecture for embodiment independent sensor-based control of manipulation, [Eskridge and Hougen 2009] propose an evolutionary approach for development of an action controller, or [Johnson and Demiris 2005] propose hierarchically coupled internal inverse and forward models for motor abstraction, to name but very few.

A integrative approach to abstraction of actions, while at the same time recognizing and classifying is presented in numerous publications, e.g., [Geib et al. 2006, Wörgötter et al. 2009], from the *paco+* project[3]. Here, so called *Object-Action-Complexes* are introduced. An object-action-complex is a combined representation of an object and the potential actions that may be performed on the object (see [Krüger et al. 2009] for a formal definition). This is due to the notion that "objects are only objects because of the actions one can execute with them" [Giuliani 2011].

Summary 2.3 – **Robot Action**

Robot actuation in principle requires

1. low-level hardware control, i.e., sending motor commands,
2. motion coordination, i.e., kinematic modeling and inverse kinematics, and
3. action abstraction, i.e., hierarchical composition from motion primitives to trajectories to complex behavior.

[3] http://paco-plus.org

2.1.3 Perception and Attention

An essential assumption of this work is, as stated before, that in general perception is a projection of transformation occurring in the environment into the sensor space of an agents perception system. This can then be used for generating invariant environment representations, proprioception, and, if associated with actuator movement, sensorimotor coordination.

Attention

It is vastly argued, that while perceiving sensory stimuli, much of the information obtained from sensory devices is irrelevant to the current task of an agent. Therefore, the system has to compute salient data features and filter input accordingly (e.g., [James 1890, Broadbent 1958, Kahneman 1973, Lavie et al. 2004]). As in this thesis a *selective visual attention* mechanism is integrated in the attention layer in close coupling with the sensor / actuator system, next also some relevant related research in the field of saliency computation, with particular focus on visual perception, is reviewed. There are two principle mechanisms involved in how an agent pays attention: (1) *bottom-up* attraction, which originates from the sensor / actuator data directly, and (2) *top-down* induction, where both reflexive and volitional feedback mechanisms affect the activation in the attention system.

There have been many approaches to computation of salient bottom-up features in a static image, e.g. [Reinagel and Zador 1999] show that high contrast regions seem to attract attention or [Kadir and Brady 2000] report that salient regions can be computed using multiscale images. [Gilles 1998] on the other hand argues that local complexity can be a measure of saliency. Also, a learning approach for visual saliency models has been proposed recently in [Kienzle et al. 2006]. Following these ideas fundamental attention attractors originating from sensory input can be either static salient features in a single frame or dynamics in the input data sequence, i.e., with consideration of temporal properties (see [Müller and Knoll 2008a, Müller and Knoll 2009a] for preliminary work and further details).

It is clear, that vision systems providing similar capabilities to the one proposed here exist. E.g., [Itti and Koch 2001, Itti et al. 1998] implement a visual attention system utilizing multiscale images to compute a saliency map. In their system a neural network selects the attended locations for detailed analysis, or, [Walther et al. 2002] use a static architecture to perform bottom-up attention based selection and attentional modulation to speed up the recognition process of their connectionist *HMAX* system, to name but a few.

In the differentiation of [Wright and Ward 2008] the visual saliency approaches mentioned above can best applied to *covert* systems. Covert attention refers to mere mental processes, where no physical tuning of the sensor towards a stimulus is applied. This is opposed to *overt* attention mechanisms, where the sensors are tuned towards an attended stimulus physically – as mostly happens in active perception systems[4].

As so far bottom-up effects were discussed, it is also worth considering the (top-down) theory of *inhibition of return*, which was shown to be plausible in human visual psychophysics, e.g., by [Posner and Cohen 1984, Cohen et al. 2004]. Here it is stated that after attending a certain area in the perceptive space inhibitory top-down effects apply to decrease activation in this area for some time interval. This is supposed to be an non-volitional, reflexive effect [Tipper and Kingstone 2005].

Also, volitional, i.e., conscious prediction effects show an influence on attention [Newman et al. 2007]. This means, to some degree, the agent controls on what to pay attention to in its input information [Corbetta 1998]. In order to do so, the agent needs to maintain a higher-level representation of input information (i.e., track an object or update an environment model). Tracking an object model by means of Lie theory (e.g., for compositional hypotheses updates) can in particular be reviewed from [Drummond and Cipolla 2002, Drummond and Cipolla 1999b, Drummond and Cipolla 1999a]. A more general, extensive survey on model-based tracking methods in the visual perception domain can on the other hand be found in [Lepetit and Fua 2005]. This process is closely correlated to the claim of the memory-prediction model [Hawkins 2004] explained earlier, where volitional control is induced by prediction (feedback) and in coherence with bottom-up attraction. Some aspects of this have also been discussed in earlier work, e.g., [Müller and Knoll 2008b].

Active Perception

Well accepted theories (e.g., [Noë 2004]) consider perception to be an active process involving the exploration of the environment properties. With *sensorimotor contingencies* [O'Regan and Noë 2001] the framework of the theory explains experimental findings, such as the inattentional or change blindness. Inspired by this idea, which is claimed to be biologically plausible, the selective attention approach can be extended.

Active perception then refers to a mechanism, which involves action in a way that

[4] Although active perception is of course also possible in a covert manner.

attention on the one hand guides action (to direct a sensor towards a region of saliency), on the other hand the action itself reveals attention in the system (e.g., [Balkenius and Hulth 1999, Hommel 2010]).

Furthermore, one can view active perception from two different perspectives, again following the attention classification scheme in [Wright and Ward 2008]. Then, covert active perception corresponds to mentally, but actively exploring the sensory input information (without visible action), in contrast to overt active perception, where attention actually drives a sensor, e.g., causes eye movements in a human(oid). Interestingly, previous work [Müller and Knoll 2011] has shown, that it is even possible to use evolutionary mechanisms to tune a covert attention selection system towards a specific task.

Summary 2.4 – **Perception and Attention**

The principles of perception considered most relevant for this work include:

❶ Perception is a projection of world transformation into sensor space and its processing.

❷ Salient features have to be computed to filter the vast amount of input information from sensors. Selective attention mechanisms propagate according to a relevance measure.

❸ Active perception systems provide means to modify sensory stimuli actively and explore the perceptive space with respect to the agent task.

In the next section the basic mathematical concept for active perception and representation of transformation by means of Lie theory, the central claim of this thesis, is briefly introduced.

2.2 Group Theory, Lie Groups and Algebras

Group theory in mathematics is a powerful tool to describe (symmetry) classes of elements and their connection to and interaction with each other. Elements in general can be anything we can possibly think of, be it physical objects, or even pure virtual ideas – basically any set of things with a common property.

In the context of this thesis, in particular a special kind of element is elaborated on: the *transformation*[5]. A transformation in mathematical notion is an *operator* T which eventually acts on some structure $\boldsymbol{x}$ representing a posture in its reference space. Applying the operator transforms the structure into another representing some different posture. Consider for example the three dimensional world as the space of reference, where any object position and orientation can be represented as a vector. Applying a transformation operator to that vector may push the object to some other location and change the orientation in the world.

Now, in mathematics, these operators can be classified by the type of action they perform on an object. One could for example easily construct a class of transformations T_d, where their common property is, that they push an object left or right

$$\boldsymbol{x} \rightarrow \boldsymbol{x}' : \begin{pmatrix} x'_1 \\ x'_2 \\ \dots \\ x'_n \end{pmatrix} = T_d \circ \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} = \begin{pmatrix} x_1 + d \\ x_2 \\ \dots \\ x_n \end{pmatrix}. \tag{2.1}$$

Of course, a group as a class of operators in the mathematical sense has some more formal properties which are explained in Section 2.2.1 and Section 2.2.2, but in general a transformation group contains operators with some property in common. One step further, a transformation group could for example comprise not only operators shifting left and right, but also for rotating an object, which would make the group gain a dimension. Then, the most important subsequent insight is, that the operators are decomposable into their *primitive prototypes* of interaction, i.e. in this case the shift and the rotation. The magnitude of a shift or a rotation can be expressed by a single parameter, a *scale* s of this primitive prototype T_p. So the actual operator in the mathematical sense can be written as a function of the scale; and the elements (transformations) in the group of operators are a composition of one or more scaled prototypes. Finding these primitive prototypes for a group is explained in great detail in Section 2.2.3.

Next we find, that in the physical world, we can never push an object to a new location in an instant, the action of pushing always takes place in a temporal context, i.e. it takes some time to move an object from position a to position b (see Figure 2.4).

The important finding here is, that a motion represented as a transformation must

[5]Note that throughout this chapter the matrix representation of transformations is used. Appendix A elaborates on advantages and disadvantages of this representation and introduces more efficient representations with respect to Lie groups and algebras for robotic applications.

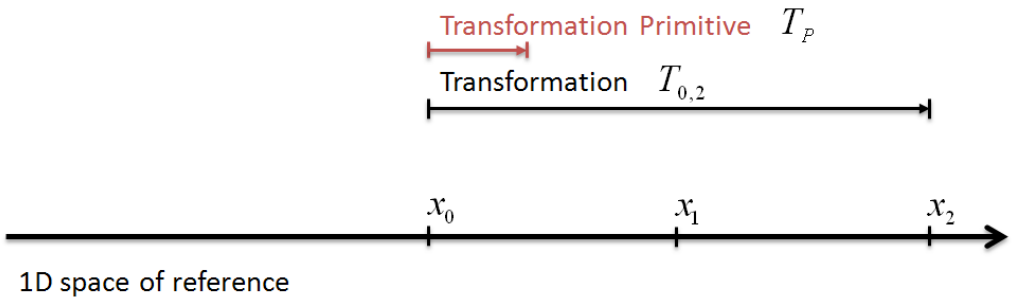

Figure 2.4 *The transformation operator as a scaled primitive in a one dimensional space of reference.*

be a function of time. Hence, with respect to FIGURE 2.4, applying a scale $s_{0,1}$ and $s_{1,2}$ to x^{t_0}

$$(s_{1,2}T_p) \circ (s_{0,1}T_p) \circ \boldsymbol{x_0}, \tag{2.2}$$

and applying a scale $s_{0,2} = s_{1,2} + s_{0,1}$

$$(s_{0,2}T_p) \circ \boldsymbol{x_0} \tag{2.3}$$

are the same in the amount they transform an object position (both move an object $\boldsymbol{x_0} \rightarrow \boldsymbol{x_2}$), but are still not identical, as they may refer to different time contexts (or different velocities respectively). Nevertheless, because time itself is continuous, also the group of transformations must be continuous and is hence called a *differential manifold* in mathematics.

Moreover, this on the other hand implies, that we can express a motion from $\boldsymbol{x_n}$ to $\boldsymbol{x_{n+1}}$ as a sequence of possibly infinitesimal increments acting always on the current position of the point object. It will be shown, that it is sufficient to consider transformations that operate only in the vicinity of the current state. A transformation that maps the current state on itself is called the *identity* transformation. SECTION 2.2.4 shows, how one can map from a differential space of scaled transformation primitives close to zero (which refers to a transformation close to the identity) into the group of continuous transformations by means of the *exponential mapping*.

2.2.1 Group Theory

Consider a set of elements of some kind S and an operator $\circ$ that combines two of the elements. The (binary) operator $\circ$ is called the *group operator*. The operator and the elements form a group $g(S, \circ)$ in the mathematical sense, if they satisfy all of the following conditions, called the *group axioms*:

Definition 2.1 – Closure *Combining an element $t_0 \in S$ with another $t_1 \in S$ by using the group operator $\circ$ results in an element t_2 of the "same kind", i.e. also being a member of the group.*

$$t_0 \circ t_1 = t_2 \tag{2.4}$$

To form a group the set S must be closed under $\circ$, or put in a more abstract way, a mapping f must exist such that $f : S \times S \mapsto S$ is correct for all $t \in S$. Furthermore, the mapping f must be associative, that is, it must satisfy Definition 2.2.

Definition 2.2 – Associativity *A mapping f being associative means, for any combination of elements $t \in S$, it does not matter, in which order the group operators are evaluated:*

$$(t_0 \circ t_1) \circ t_2 = t_0 \circ (t_1 \circ t_2) \tag{2.5}$$

This property does not at all state, that operations are commutative. In fact, in general it is absolutely not guaranteed that the order of elements may be exchanged without effects.

To be a group, the set S with the operator $\circ$ must in addition to closure and associativity have an unique identity element (see Definition 2.3) and an inverse element $t^{-1} \in S$ for each $t \in S$ (see Definition 2.4).

Definition 2.3 – Identity Element *The identity element t_{id} is an element that does not change any other element $t \in S$ (maps t onto itself) when combined with the group operator $\circ$ (NB. this operation is commutative):*

$$\forall t : t \circ t_{id} = t_{id} \circ t = t \tag{2.6}$$

Definition 2.4 – Inverse Element *The inverse element t^{-1} for each $t \in S$ is an element $\in S$ such that the change to any other element $t_n \in S$, can be reversed:*

$$t_0 \circ t_1 = t_n \quad \Rightarrow \quad t_0 = t_1^{-1} \circ t_n \tag{2.7}$$

Put simple, Definition 2.4 states, that combining an element with its inverse must result in the identity:

$$t \circ t^{-1} = t^{-1} \circ t = t_{id} \tag{2.8}$$

The above is a general, but rather abstract definition of a group $g(S, \circ)$, that can be found in most textbooks of mathematics, e.g., [Selig 2005]. For clarification an (almost trivial) example follows without proof.

A very simple group is $g(\mathbb{Z}, +)$, where the group operator $\circ$ is $+$ and the set of elements contains all (signed) integers.

1. The identity is 0 here, as

$$\forall z \in \mathbb{Z} : z + 0 = 0 + z = z \tag{2.9}$$

2. The inverse is $-z$, as

$$\forall z \in \mathbb{Z} : z + (-z) = 0 \tag{2.10}$$

3. The order of operator evaluations is arbitrary, as

$$\forall z_0, z_1, z_2 \in \mathbb{Z} : (z_0 + z_1) + z_2 = z_0 + (z_1 + z_2) \tag{2.11}$$

4. Combining any element z_1 with any other element z_2 obviously creates an element from within $\mathbb{Z}$ (closure), as $\mathbb{Z}$ spans an interval $]-\infty, \ldots, -1, 0, 1, \ldots, \infty[$.

This group is actually also commutative, but we do not care about this property here, as group elements in general do not have to commute.

2.2.2 Lie Groups

Being familiar with a mathematical group in general this section shows, how the group definitions can be specialized for a Lie group $g_L(S, \circ)$, which is sometimes also called *continuous transformation group*. Hence, there are two important aspects to be discussed: first, the concept of a *group of transformations* and second, the concept of *continuity*.

Concerning transformation groups, we have to concretize our definition from the last section. There, a group was just a set of elements of arbitrary kind, for example integers above. Now, we require the elements to be *transformations*. A transformation is an operator itself, which should not bother to much in this section, but be kept in mind for discussions below.

Transformations always act on, i.e., "transform", something. Concerning applications in robotics or computer vision, we normally want a transformation T to act on a vector of real numbers $\boldsymbol{v}$, e.g., a tuple (2D) or triple (3D) of coordinates and 2D or 3D orientation parameters. So a transformation applied to some coordinates results in some new coordinates[6].

[6] This is not to mix up with the closure axiom from Definition 2.1, as for a group of transformations, algebraic closure requires a combination of transformations to result in a novel transformation, not a vector.

Furthermore, we focus on transformations $T_x = T(x)$ that can be parametrized by a single real number x. As we will see below, multivariate transformation functions may be decoupled into several single-parameter transformations for treated robotic applications.

For example, considering the action of a transformation subject to x on some point in space, this can be written as

$$\boldsymbol{v'} = T(x)\boldsymbol{v}. \tag{2.12}$$

One important property has to be noted here: the identity transform T_{id} corresponds to a parameter $x = 0$ in all cases contemplated below,

$$T(0) = T_{id}. \tag{2.13}$$

Whether the group axioms hold for a set of transformations (without an extensive proof), one can easily check.

- *Closure*
  A transformation can be combined with another transformation, the result will always be a transformation.
- *Associativity*
  It does not matter, in which order transformations are combined binarily, if only their sequential order is kept.
- *Identity Element*
  An identity element exists by definition, $T(0)$.
- *Inverse Element*
  An inverse transformation can be constructed easily, normally it can be written $T(x)^{-1} = T(-x)$.

Continuous transformation groups comprise a set of transformations that satisfy one additional property: this set is continuous. These groups are today also named *Lie groups* after the ingenious Norwegian mathematician Sophus Lie, who first investigated their special properties [Lie 1872].

A *transformation* is called continuous, if its action on a coordinate / orientation parameter set can be made infinitesimally small. That is e.g., for a translation, one can always cut the displacement expressed by a transformation into halves until it vanishes. A *transformation group* is continuous, if the function $T(x)$ maps onto the set of transformations S^T smoothly. I.e., whenever x is varied infinitesimally

$x_1 = x_0 \pm \epsilon$ with $\epsilon \to 0$ and fed into $T(x)$, the result transformation T_{x_1} describes an infinitesimal variation of T_{x_0}. In the linear case this is equivalent to writing:

$$x \propto T(x) \tag{2.14}$$

Summary 2.5 – **Lie Groups**

Briefly, again the essential properties of a Lie group are:

- The "normal" group axioms: **closure**, **associativity**, existence of an **identity** element and an **inverse** element for each group member.
- Lie groups are **transformation groups**, i.e., all members are transformations.
- **Continuity**, which means the group has the properties of a differential manifold, i.e., for each two transformations we can find another one from the same Lie group located "in between" the two others.

As a simple example, think of 1D space denoted by a homogeneous vector $\boldsymbol{v} = (v_0, 1)^T$ (comprising a single real coordinate $v_0 \in \mathbb{R}$). Furthermore, consider a certain translation T_x to be defined by a function

$$T_x = T(x) = \begin{pmatrix} 1 & x \\ 0 & 1 \end{pmatrix} \tag{2.15}$$

Here, a translation T_x moves the point's coordinate by x. We can easily see, that variations of x are proportional to variations of the transformation T_x, because the transformation function $T(x)$ is linear:

$$\boldsymbol{v'} = T_x \boldsymbol{v} = \begin{pmatrix} v_0 + x \\ 1 \end{pmatrix} \tag{2.16}$$

Thus we can see, that this linear $T(x)$ is continuous and hence differentiable. Here in fact the derivative

$$\frac{d}{dx} T(x) = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \tag{2.17}$$

is also constant. In general, we can now define a Lie group.

Definition 2.5 – Lie Group *$g_L(S^T, \circ)$ is a continuous transformation group (a Lie group), whenever the function $T(x)$ is differentiable, i.e., $T(x) : \mathbb{R} \mapsto S^T$ is a differential mapping.*

Or put another way, whenever the set of transformations S^T is a differential manifold then S^T, together with the group operator $\circ$ forms a Lie group. Finally, we have listed the ingredients of a Lie group $g_L(S^T, \circ)$ and may now continue with an introduction to *Lie algebras*.

2.2.3 Lie Algebras

Informally, we can say, that the Lie algebra $\boldsymbol{A}$ over a Lie group allows for efficient (locally coherent) construction of elements of a Lie group close to the identity transformation. This means, one can view a Lie algebra element as a general velocity construct, valid at the identity, from which an increment on the current state can be created with respect to a sufficiently small time step. This is a multivariate analogon to the simple $f(x) + f'(x) \cdot \delta x \approx f(x + \delta x)$ of undergraduate analysis, where $f'(x)$ corresponds to a derivative (e.g. a velocity), and which obviously is only valid for small δx.

Each element of a Lie algebra can be written as a vector of parameters from which a transformation (an element of the group) can be composed using infinitesimal primitive prototypes. While Section 2.2.4 describes how this can be done using the exponential map, this section beforehand shows, how the Lie algebra and corresponding prototypes, also called *generators* can be derived from a Lie group.

For a moment, consider the Lie group for translations in 2D space. Here, the transformations T act on 2D homogeneous coordinates $\boldsymbol{v} = (v_x, v_y, 1)^T$ with $v_x, v_y \in \mathbb{R}$, i.e., the x and the y values.

$$\boldsymbol{v'} = T\boldsymbol{v}$$

Each element of the group $T \in S^T$ is a parametrized ("scaled") composition of transformations from the orthonormal basis of the group, i.e., of translations in x-direction and/or y-direction

$$T = T_x(a) \cdot T_y(b), \tag{2.18}$$

where a and b are the scales in x- and y-direction and the orthonormal basis transformations subject to the single parameter can then in this example be written as a

function of the parameters a and b as follows:

$$T_x(a) = \begin{pmatrix} 1 & 0 & a \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad T_y(b) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & b \\ 0 & 0 & 1 \end{pmatrix} \tag{2.19}$$

So the group elements $T \in S^T$ can also be written as a multi-variate function of the parameters

$$T(a, b) = T_x(a) \cdot T_y(b). \tag{2.20}$$

But how can we describe all elements of the group close to the identity in general and efficiently? Here, luckily we can use the differential property of a Lie group. Thus, if we know the single parameter transformations, of which the group elements can be composed (w.r.t. the orthonormal basis of the group), we can differentiate and set the parameter to zero. E.g., for $T_x(a)$ from Equation (2.19) above this yields

$$G_x = \frac{\partial}{\partial a} T(a, b)|_{a=0} = \frac{d}{da} T_x(a)|_{a=0} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}. \tag{2.21}$$

Within the scope of Lie systems, this differential G_x is normally called *infinitesimal generator*, because it is able to "generate" small transformation in one "direction" of the orthonormal basis of the group, as we will see below.

For the translation group mentioned above, setting $a = 0$ does not change anything in the differential of Equation (2.21). But now consider for example the group of rotation transformations for a 2D point with angle θ around the z-axis in homogeneous 2D, represented as a 3×3 matrix:

$$T_R(\theta) = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.22}$$

Following the above approach, the generator G_R of the Lie algebra for that transformation group can be expressed as:

$$G_R = \frac{d}{d\theta} T_R(\theta)|_{\theta=0} = \begin{pmatrix} -\sin\theta & \cos\theta & 0 \\ -\cos\theta & -\sin\theta & 0 \\ 0 & 0 & 0 \end{pmatrix} \Bigg|_{\theta=0} = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{2.23}$$

Here we can see, that setting $\theta = 0$ clearly makes a difference and yields a rather simple, neat expression for the generator G_R. The steps necessary to extract the infinitesimal generators are summarized in Summary 2.6.

Summary 2.6 – **Infinitesimal Generators**

- Consider the a transformation $T \in S^T$ subject to e.g. two parameters $T(a, b)$. The (partial) derivative with respect to one "direction" then is:

$$\frac{\partial}{\partial a} T(a, b) = \frac{d}{da} T_x(a)$$

- The infinitesimal generator for transformation subject to a then is the derivative, evaluated at the identity $a = 0$:

$$G = \frac{\partial}{\partial a} T(a, b)|_{a=0} = \frac{d}{da} T(a)|_{a=0}$$

Definition 2.6 – Lie Algebra *A Lie algebra $\boldsymbol{A}$ is a special case of a vector space. It has a binary operator $[\cdot, \cdot] : \boldsymbol{A} \times \boldsymbol{A} \mapsto \boldsymbol{A}$ called the Lie bracket[7] $[X, Y] = XY - YX$ following additional constraints, i.e. bilinearity, alternation and the Jacobi identity.*

A Lie algebra[8] is a vector space where each element has the dimension corresponding to the number of orthonormal basis transformations (the transformation primitives). For the above example S^T of 2D translations, elements of the Lie algebra $\boldsymbol{A}$ are then two dimensional vectors $\boldsymbol{s} \in \mathbb{R}^2$. The entries of a vector $\boldsymbol{s}$ are the *scales* s that can be multiplied to the inifinitesimal generators. So, a generator G of the Lie algebra over the group $g(S^T, \circ)$ may be scaled by a parameter s.

For G_R defined in Equation (2.23), we can then write a corresponding *generator function* $G_R(s)$:

$$G_R(s) = sG_R = \begin{pmatrix} 0 & s & 0 \\ -s & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{2.24}$$

Considering this, the Lie algebra spans a tangent vector field of the corresponding Lie group at the identity. This is analogous to multivariate functions, where the (partial) derivatives also define a tangent vector field at some point, only that the

[7] Here, the concept of *Lie brackets* is not explained in greater detail. An understandable definition can be reviewed for example from [Selig 2005, pp. 57].

[8] The term Lie algebra, formerly called "infinitesimal group", was introduced in the 1930's by Hermann Weyl in honour of Sophus Lie [Weyl 1939].

Lie algebra elements span this field merely at the identity transformation.

As an example, we revise the easiest case of one-dimensional space (Section 2.2.2). The only thing one can do with a coordinate is to increment or decrement the value. Thus, we again design the continuous transformation function to be $T(x) = \left(\begin{smallmatrix} 1 & x \\ 0 & 1 \end{smallmatrix}\right)$ with $x \in \mathbb{R}$, so $T(0) = T_{id}$.

For this group, the Lie algebra $\boldsymbol{A}$ only has one generator G_T,

$$G_T = \frac{d}{dx} T(x)|_{x=0} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \tag{2.25}$$

so in this case, with scaling s, $G_T(s) = \left(\begin{smallmatrix} 0 & s \\ 0 & 0 \end{smallmatrix}\right)$ defines the tangent, as it "knows" the direction. The Lie algebra $\boldsymbol{A}$ is then defined as a one-dimensional vector space

$$\boldsymbol{A} = \{\boldsymbol{s} | \boldsymbol{s} = (s), s \in \mathbb{R}\} \tag{2.26}$$

Next, for a more interesting example think of a certain group with three different basic single-parameter transformations, for example one for each translations in x- and y-directions and one for rotations about the z-axis in 2D. We have already listed the three generators (a orthonormal basis of the group) for that in Equation (2.21) and Equation (2.23):

$$G_x = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad G_y = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \quad G_R = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{2.27}$$

The elements of the Lie algebra are vectors $\boldsymbol{s} \in \mathbb{R}^3$ then. Each of the entries of a vector specifies a parameter for one of the generator functions, e.g.,

$$G_x(s_x) = \begin{pmatrix} 0 & 0 & s_x \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad G_y(s_y) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & s_y \\ 0 & 0 & 0 \end{pmatrix}, \quad G_R(s_R) = \begin{pmatrix} 0 & s_R & 0 \\ -s_R & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \tag{2.28}$$

so the algebra can be written as vector space of three scales each,

$$A = \{\boldsymbol{s} | \boldsymbol{s} = (s_x, s_y, s_R)^T \text{ with } s_x, s_y, s_R \in \mathbb{R}\}. \tag{2.29}$$

In this section it was shown, how the infinitesimal generators from a Lie group can be obtained, and how the Lie algebra and their elements can be derived (see Summary 2.7). But how is it possible to obtain an element of the Lie group from its corresponding element in the Lie algebra? This will be explained in the following section.

Summary 2.7 – **Lie Algebra**

- For each Lie group, we can find a corresponding Lie algebra - a tangent vector space at the identity element of the group.
- To specify the elements of the Lie algebra, we first have to define the orthonormal basis transformations of the group and find their (infinitesimal) **generators**.
- The corresponding **generator functions** then apply a scale parameter to each generator.
- The elements of the Lie algebra are simply vectors of scale parameters for these generators.

2.2.4 Exponential Mapping

In this section the concept of *exponential mapping* is introduced. Resuming the explanations from the last section, where it was shown how to construct the Lie algebra for a Lie group, here it is investigated, how one can find an element of the Lie group corresponding to an element in the algebra. As the name "exponential mapping" already suggests, the approach to map back from the Lie algebra to the Lie group includes applying the exponential function e^x or $\exp(x)$.

Although, the exponential function has some other characteristic properties, we do particularly make use of one to explain the exponential mapping:

$$e^x = \lim_{n\to\infty} \left(1 + \frac{x}{n}\right)^n \tag{2.30}$$

As an example, we again examine one-dimensional space from above. As stated before, the differentiable transformation function $T(x)$ for $T \in S^T$ and the corresponding generator G_T are

$$T(x) = \begin{pmatrix} 1 & x \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad G_T = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \quad \text{with} \quad T_{id} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{2.31}$$

Regarding the above and $T(0) = T_{id}$, we may infer and immediately check, that a

translation by x on a vector $\boldsymbol{v} = (v_0, 1)^T$ can be written as

$$\boldsymbol{w} = \begin{pmatrix} w_0 \\ 1 \end{pmatrix} = \begin{pmatrix} v_0 + x \\ 1 \end{pmatrix} = T(x)\boldsymbol{v} = (T_{id} + xG_T)\,\boldsymbol{v}. \tag{2.32}$$

Moreover, now consider, we do not want the whole translation at once, but split it into n equal parts $\forall i, j, \in n : |x_i| = |x_j|$, maybe because of a velocity constraint, that only allows for a certain increment per timestep. For translations one after the other, we can then write

$$\begin{aligned} \boldsymbol{w} &= T\,(x)\,\boldsymbol{v} = \begin{pmatrix} 1 & x \\ 0 & 1 \end{pmatrix} \boldsymbol{v} = \begin{pmatrix} 1 & \frac{x}{n} \\ 0 & 1 \end{pmatrix} \cdots \begin{pmatrix} 1 & \frac{x}{n} \\ 0 & 1 \end{pmatrix} \boldsymbol{v} \\ &= T\left(\frac{x}{n}\right)^n \boldsymbol{v} = \left(T_{id} + \frac{x}{n} G_T\right)^n \boldsymbol{v}, \end{aligned} \tag{2.33}$$

where we compose the final translation transformation by applying n parts of $T(\frac{x}{n})$ one after the other, so we see

$$T(x) = \left(T_{id} + \frac{x}{n} G_T\right)^n. \tag{2.34}$$

Interestingly, this looks almost like the definition of the exponential function e^x from Equation (2.30). One aspect left is to verify, that $T_{id} = 1$, which is perfectly legitimate, as we know, that $T_{id}\boldsymbol{v} = 1 \cdot \boldsymbol{v}$.

As the last step towards the exponential mapping, we set $n \to \infty$ and finally write down the mapping equation:

$$T(x) = \lim_{n \to \infty} \left(T_{id} + \frac{x}{n} G_T\right)^n = e^{xG_T} \tag{2.35}$$

Thus we can derive all elements S^T of our one-dimensional Lie group by defining the generator function $G_T(x) = xG_T$ and scaling G_T systematically

$$S^T = \left\{ e^{G_T(x)} \,\middle|\, x \in \mathbb{R} \right\} \tag{2.36}$$

The approach is analogous for higher dimensional spaces and multiple generators, so we skip an extensive proof (see e.g. [Samelson 1990]) and only provide the rough idea here. A caveat within these considerations is, for multiple generators a property of the exponential function (and in fact a law of exponentiation in $\mathbb{R}$) is not guaranteed:

$$\exp(x)\exp(y) = \exp(x + y). \tag{2.37}$$

This relation would only hold if the Lie bracket $[\cdot,\cdot]$ of the algebra $\boldsymbol{A}$ always evaluated to zero $[A_1, A_2] = A_1A_2 - A_2A_1 = 0$, i.e. the algebra is commutative[9]. Nevertheless, it is possible to chain transformations in the following way.

Think of a certain group with m different basic single-parameter transformations. In the example from Section 2.2.3 one for each translations in x- and y-directions and one for rotations about the z-axis in 2D. All elements of that continuous group (the transformations) can then be composed by

$$T = T_x \cdot T_y \cdot T_R. \tag{2.38}$$

We then have multiple generators $G_i \in \{G_1, \ldots, G_m\}$ (with $m = 3$ for the above). Therefore, we need an m-dimensional scale vector $\boldsymbol{s} \in \mathbb{R}^m$. Next, we apply the scale vector to the generators and chain the resulting base transformations to generate the group elements $T(\boldsymbol{s}) \in S^T$:

$$\begin{aligned} T(\boldsymbol{s}) &= T(s_1)T(s_2)\ldots T(s_m) \\ &= e^{s_1G_1}e^{s_2G_2}\ldots e^{s_mG_m} \end{aligned} \tag{2.39}$$

With the exponential $e(x) = \exp(x)$, we can in general write this compactly as

$$T(\boldsymbol{s}) = \prod_{i=1}^{m} T_i(s_i) = \prod_{i=1}^{m} \exp\left(s_iG_i\right). \tag{2.40}$$

One more property used frequently in this work is *additivity* in the scales of a specific generator G_i. It is also given as is without the corresponding proof derived e.g. in [Samelson 1990], as

$$\exp(s_i^0G_i)\exp(s_i^1G_i) = \exp((s_i^0 + s_i^1)G_i). \tag{2.41}$$

Here, we do not elaborate on how to compute the exponential on matrices M such that $M = \exp(s_iG_i)$. A multitude of generic approaches for this purpose have been proposed [Moler and Van Loan 1978, Moler and Van Loan 2003], including analytical exponentials for special cases and the matrix exponential as a limit of powers, a sum of powers, via the Laplace transform, or using the "scaling and squaring" method [Lawson 1967, Higham 2004] for the more general ones. This is due to the context of this thesis, where group theory is applied to robotic applications. In that context, most relevant groups are the special Euclidian group $SE(3)$ and the special orthogonal group $SO(3)$ introduced later, for which closed forms of the exponential can be derived [Govindu 2003, Agrawal 2005].

[9] Details on Lie brackets and (non-)commutative algebra properties are discussed in Section 3.2.2.

Now one can easily see the advantage of a transformation design, such that $T(0) = T_{id}$: a scale $s_i = 0$ in the corresponding transformation $T_i(s_i)$ causes no change on the result. Hence, especially for the algebra element $\boldsymbol{s^0} = (0, \ldots, 0)^T$, we state

$$T(\boldsymbol{s^0}) = \prod_{i=1}^{m} \exp\left(0 \cdot G_i\right) = \prod_{i=1}^{m} \exp \begin{pmatrix} 0 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & 0 \end{pmatrix} = T_{id}, \tag{2.42}$$

i.e., for all zero parameters (the zero element in the Lie algebra) we map to the identity element in the Lie group, the transformation T_{id}.

Summary 2.8 – **Exponential Mapping**

- The **exponential map** allows for transferring elements of a Lie algebra close to the identity back into the corresponding Lie group.
- Consider a linear combination of n infinitesimal generators G, the corresponding transformation T would be

$$T = \prod_{i=1}^{m} \exp\left(s_i G_i\right) \quad \text{with} \quad \boldsymbol{s} \in \mathbb{R}^n.$$

- The zero algebra element $(0, \ldots, 0)^T$ maps to the identity transformation

$$T(\mathbf{0}) = T_{id}.$$

2.3 Lie Theory for Robotics

Up to this point, 3D transformations as elements of a Lie group, constructing the corresponding elements in the Lie algebra, as well as mappings between the two have been explained.

The next sections introduce application of these mathematical structures to serial mechanical systems and robotics in general. First, Lie algebras and their application to simple mechanical structures, i.e., revolute joints, are discussed in Section 2.3.1 and Section 2.3.2. Then, as in reality robots comprise more than one joint, it is

examined, how the results can be extended to advanced robotic systems. Furthermore, Section 2.3.3 elaborates on such a system's dynamics, i.e., how group theory can be used to compute velocities and Jacobians elegantly for serial manipulators. A solution to the important problem of generating smooth paths and trajectories in realtime for multi-joint serial manipulators is covered in Chapter 4, where also original work on further generalizing these concepts to perception and sensorimotor association systems is presented.

2.3.1 Kinematics and Revolute Joints

We live in a three-dimensional world. Postures in 3D space have at least six parameters, three for position and three for orientation, i.e., six degrees of freedom. Now the problem of *forward kinematics* states the problem of finding the pose of a (serial) robot end-effector, for example of a gripper mounted to the robot in world coordinates corresponding to some angle configuration of the robot's joints.

Looking at the design of current robots, we mainly find two different types of joints: those applying a translation along one axis, called *prismatic* joints and those applying a rotation around an axis, called *revolute* joints. Without further details, e.g., on *Releaux's lower pairs* [Reuleaux 1875], we state that revolute joints are sufficient to cover the whole scope of motion in 3D space. For example, Figure 2.5 shows, how a prismatic joint can be substituted with a combination of revolute ones. Thus, the following paragraphs will focus on elaborations on revolute joints.

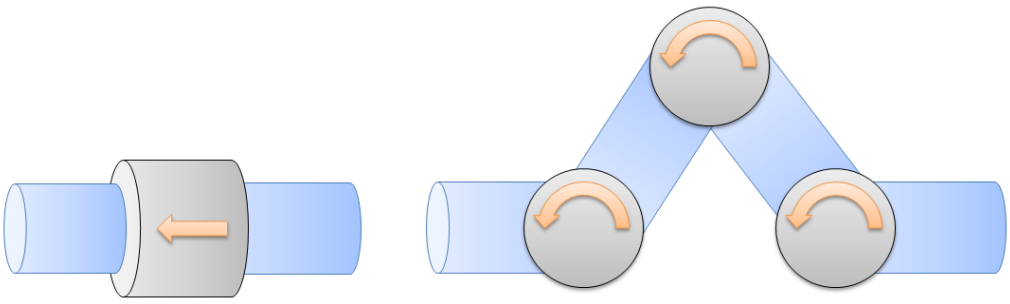

Figure 2.5 *Simple substitution of a prismatic joint (left) with revolute joints (right).*

Consider a joint angle configuration[10] $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_n)$ of a robot at some point in time. In theory $n = 6$, i.e., six sequential revolute joints are sufficient to represent six degrees of freedom as long as they provide a basis of $SE(3)$. However, the

[10] In literature, mostly the joint angle configuration is denoted by a vector $\boldsymbol{q}$. In this work however, $\boldsymbol{\theta}$ is used instead to avoid confusion with the quaternion notation for rotations.

following approach also works for more than six joints. The benefit of a redundant system with $n > 6$ would for example be a reduction of singular configurations (see [Boudreau and Podhorodeski 2010] for a more extensive discussion on the topic).

The most wide-spread, quasi standard approach to compute the end-effector pose uses the *Denavit-Hartenberg* (DH) parameter convention [Denavit and Hartenberg 1955]. The convention defines a way to specify the reference frame of a joint j_n in terms of its predecessor j_{n-1}. Details on the Denavit-Hartenberg convention are beyond the scope of this work and can be reviewed from every ordinary textbook on robotics.

2.3.2 Lie Algebras for a Joint

As stated before, regarding a robot with revolute joints, in general we need at least six sequential joints to be able to reach any pose in space[11], because every joint only provides for one degree of freedom. Manipulators with six serial revolute joints are commonly also called 6R manipulators, e.g., [Manocha and Canny 1994].

In order to utilize the framework of group theory for robotic applications, we must first investigate a single revolute joint (see Figure 2.6), before we integrate the approach for multiple sequential joints and derive efficient exponential formulae for end-effector pose, joint velocities, and so on in the next sections.

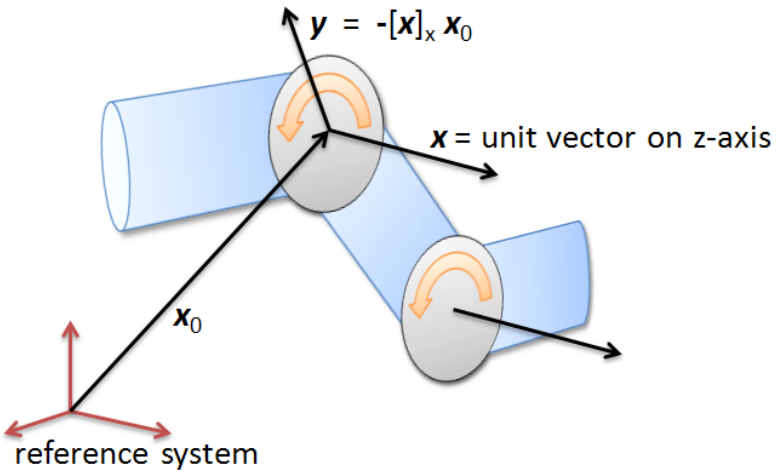

Figure 2.6 *A revolute joint and its parameters.*

Consider a function that specifies motion of an object attached to a single joint, e.g., a mechanical gripper. Apart from the constant robot configuration, i.e., the length

[11] This general statement has some limitations, i.e., the system may be exposed to a singularity if certain conditions are met. In [Hollerbach 1985] this issue of 6R manipulators is discussed in great detail.

of the gripper and the position of the joint relative to the world-origin, for example computed using the *Denavit-Hartenberg-Convention* [Denavit and Hartenberg 1955], there is only a single parameter to influence the pose of the tool-tip: the rotation angle θ of the joint.

Comparing this insight to what is derived in Section A.3, i.e., the concept of screw motions for representing transformations in $SE(3)$, one can immediately see the parallels. We only need to find screw motion parameters, such that the axis (the unit vector $\boldsymbol{x}$ of the screw motion) is aligned with the axis of rotation of the joint and that it passes through the origin of the joint coordinate frame. In other words, we need to find a Lie algebra element $\boldsymbol{s} = (s_1, s_2, s_3, s_4, s_5, s_6)$ specific to the joint of our robot, such that

$$T_s(\theta) = \exp(\theta G_s) \quad \text{with} \quad G_s = \sum_{i=1}^{6} s_i G_i. \tag{2.43}$$

G_s is in general called *screw*. The vector $\boldsymbol{s}$ of parameters defines a *twist* [Ball 1876]. As always, the infinitesimal generators G_i form an orthonormal basis of the six-dimensional vector space $se(3)$. As shown in Appendix A, we can represent these generators as 4×4 matrices. For rotations we thus use the three generators from Equation (A.42), for translations, we take the generators from Equation (A.41).

Given $\boldsymbol{s}$ is a composition of angular velocity parameters given by $\boldsymbol{x} \in \mathbb{R}^3$ and linear velocity parameters $\boldsymbol{y} \in \mathbb{R}^3$,

$$\boldsymbol{s} = (\boldsymbol{x}, \boldsymbol{y}) = (x_1, x_2, x_3, y_1, y_2, y_3), \tag{2.44}$$

we can write the screw $G_{\boldsymbol{s}}$, a linear combination of the G_i, compactly as

$$G_{\boldsymbol{s}} = \sum_{i=1}^{6} s_i G_i = \begin{pmatrix} [\boldsymbol{x}]_\times & \boldsymbol{y}^T \\ 0 & 0 \end{pmatrix} \tag{2.45}$$

Finding a such algebra element $\boldsymbol{s} \in \mathbb{R}^6$ of $se(3)$, i.e., the twist for the first revolute joint is straightforward, as all constraints known from the robot geometry can be applied directly.

Given is the joint location and orientation from robot geometry specified by some transform T, such that the z-axis of the joint coordinate frame is aligned with the axis of rotation (see Figure 2.6)[12].

[12] Following the *Denavit-Hartenberg* convention [Denavit and Hartenberg 1955], the z-axis is taken to be the rotation axis of a joint. In general of course, the approach is analoguous for alignment of the screw axis with a different joint rotation axis.

1. The vector of rotation parameters $\boldsymbol{x}$, i.e., the axis of rotation of the screw particular to our joint, is easy to obtain. We know, it has to be a unit vector $|\boldsymbol{x}| = 1$. Furthermore, we know that the z-axis in the joint coordinate frame is the axis of rotation of the screw motion. Thus, we can transform a point on the z-axis of our world frame $\boldsymbol{p}_{world} = (0, 0, 1, 1)$ of unit length into the joint frame

$$(\boldsymbol{x_p}, 1)^T = T\boldsymbol{p}_{world}^T. \tag{2.46}$$

   To define the unit vector of the screw axis $\boldsymbol{x}$, we simply subtract the world coordinate of the joint origin given by $(\boldsymbol{x_o}, 1)^T = T \cdot (0, 0, 0, 1)^T$ from $\boldsymbol{x_p}$

$$\boldsymbol{x} = \boldsymbol{x_p} - \boldsymbol{x_o}. \tag{2.47}$$

2. The translation parameters, i.e. a point on the rotation axis of the screw motion can be chosen as the origin in joint coordinates $(0, 0, 0, 1)^T$ and in world coordinates as $(\boldsymbol{x_o}, 1)^T = T \cdot (0, 0, 0, 1)^T$, as stated above. In principle (i.e. from Equation (A.48)) we know, if we want to describe screw motions about an axis not passing through the origin, we have to translate the axis (i.e. a point on the axis) into the origin and translate back afterwards. Thus, the translational part of the screw motion T_S was given in terms of $\boldsymbol{x}$, θ, p, and a point on the axis $\boldsymbol{x_o}$ as

$$\boldsymbol{y} = \frac{\theta p}{2\pi}\boldsymbol{x} + (I_3 - R(\theta, \boldsymbol{x}))\, \boldsymbol{x_o}. \tag{2.48}$$

   Luckily, a revolute joint does not apply any translation to the screw motion, so the pitch p specifying the displacement along the axis $\boldsymbol{x}$ is $p = 0$. The above then becomes

$$\boldsymbol{y} = (I_3 - R(\theta, \boldsymbol{x}))\, \boldsymbol{x_o} = (I_3 - \exp(\theta[\boldsymbol{x}]_\times))\, \boldsymbol{x_o}, \tag{2.49}$$

   as derived in Equation (A.46). Finally, we have to get back from $SE(3)$ to the Lie algebra $se(3)$, so we need to find the differential part corresponding to the translation part of the above screw motion. Differentiating with respect to θ, according to the chain rule gives

$$\boldsymbol{y} = \frac{d}{d\theta}\big(I_3 - \exp(\theta[\boldsymbol{x}]_\times)\big)\boldsymbol{x_o} = -[\boldsymbol{x}]_\times \exp(\theta[\boldsymbol{x}]_\times)\boldsymbol{x_o} \tag{2.50}$$

   and evaluating for $\theta = 0$ clearly gives

$$\boldsymbol{y} = -[\boldsymbol{x}]_\times \exp(\boldsymbol{0}_{3\times3})\boldsymbol{x_o} = -[\boldsymbol{x}]_\times I_3 \boldsymbol{x_o} \tag{2.51}$$

So essentially, we can express the rotation $R(\theta, \boldsymbol{x})$ with its generator $[\boldsymbol{x}]_\times$, the identity I_3 vanishes and we are able to simplify to

$$\boldsymbol{y} = -[\boldsymbol{x}]_\times \boldsymbol{x_o} = \boldsymbol{x_o} \times \boldsymbol{x}, \tag{2.52}$$

because the cross-product is anti-commutative.

3. At the end, the screw for out joint can be composed according to Equation (2.44) and Equation (2.45) from

$$\boldsymbol{s} = (\boldsymbol{x}, \boldsymbol{y}) = (s_1, s_2, s_3, s_4, s_5, s_6), \tag{2.53}$$

and we can write it

$$G_{\boldsymbol{s}} = \sum_{i=1}^{6} s_i G_i = \begin{pmatrix} [\boldsymbol{x}]_\times & \boldsymbol{y}^T \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -s_3 & s_2 & s_4 \\ s_3 & 0 & -s_1 & s_5 \\ -s_2 & s_1 & 0 & s_6 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \tag{2.54}$$

where $\exp(G_{\boldsymbol{s}})$ describes the infinitesimal posture variation conducted by the joint.

$G_{\boldsymbol{s}}$ is sometimes also called a *joint screw* for a screw motion T_S [Selig 2005]. Concluding this section, we have now derived a formalism to specify any $SE(3)$-transformation conducted by a revolute joint by simply applying a single rotation parameter, i.e. scaling the screw $G_{\boldsymbol{s}}$ specified by the Lie algebra element $\boldsymbol{s}$ with θ, and performing the exponential mapping

$$T_S(\theta) = \exp\left(\theta \sum_{i=1}^{6} s_i G_i\right) = \exp(\theta G_{\boldsymbol{s}}). \tag{2.55}$$

The above holds for the transformation undergone by a point in the coordinate frame of a revolute joint. Again, the essential finding is what was intuitively expected from the beginning: given geometric constraints, i.e., the joint position and orientation coded into $\boldsymbol{s} = (\boldsymbol{x}, \boldsymbol{y})$, rotating the joint by an angle θ corresponds to the transformation

$$T_S(\theta) = \exp\begin{pmatrix} 0 & -\theta s_3 & \theta s_2 & \theta s_4 \\ \theta s_3 & 0 & -\theta s_1 & \theta s_5 \\ -\theta s_2 & \theta s_1 & 0 & \theta s_6 \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{2.56}$$

Due to the differential properties of the mathematical framework, setting $\theta = 0$ results in the identity transformation T_{id}, i.e., it is only necessary to compute with

incremental (compositional) θ close to zero. This leads to numerical stability, as opposed to taking the absolute rotation angle of the revolute joint into account directly – a major improvement to the classic approach described in [DENAVIT AND HARTENBERG 1955]. Also, as the computation depends merely on the current configuration, it arbitrates from previous configurations and is thus capable to define the effect of a possible joint angle variation, which is very useful for computation of Jacobians. This and more obvious advantages considering this method appear when investigating a sequence of revolute joints, as common in todays industrial robots.

Summary 2.9 – **Lie Theory for Revolute Joints**

Considering revolute joints, the following terminology applies

- T_S refers to a transformation in the Lie group $SE(3)$, the **screw motion**.
- A Lie algebra element in $se(3)$ denoted by $\boldsymbol{s}$ is called a **twist**. It is a vector with six components, where $\boldsymbol{x} \in \mathbb{R}^3$ refers to the axis of rotation and $\boldsymbol{y} \in \mathbb{R}^3$ refers to the moment of the displacement from the origin and the axis

$$\boldsymbol{s} = (\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}, \boldsymbol{x_o} \times \boldsymbol{x}).$$

- $G_{\boldsymbol{s}}$ is called a **motion screw** or **joint screw**. It is composed of the algebra element $\boldsymbol{s}$ and the infinitesimal basis generators of $SE(3)$ given in EQUATION (A.41) and EQUATION (A.42),

$$G_{\boldsymbol{s}} = \sum_{i=1}^{6} s_i G_i = \begin{pmatrix} [\boldsymbol{x}]_\times & \boldsymbol{y}^T \\ 0 & 0 \end{pmatrix}.$$

- The exponential mapping and a rotation angle θ, the "scale", relate a twist $\boldsymbol{s}$ to the corresponding transformation in $SE(3)$

$$T_S(\theta) = \exp(\theta G_{\boldsymbol{s}}).$$

2.3.3 Compositional Mechanics and Dynamics

Up to this point, the advantage of using the differential parts of a transformation, i.e., using Lie algebra elements of $se(3)$ instead of using the transformations of $SE(3)$

directly have only been touched insufficiently.

As stated at the end of Section 2.3.2, one advantage of using Lie algebras is that it is not necessary to compute with (sometimes large) absolute values. As the elements of the Lie group, i.e., the transformations, become the identity whenever the scaling parameters of the Lie algebra elements (i.e., the screw parameters) are zero, this leads to numerical stability.

A second significant advantage is, that it simplifies the computation of derivatives with respect to time, i.e., velocities and robot *Jacobians*, and third, adopting group theory to resolve the robot kinematics question can help avoiding singularities.

2.3.4 Serial Manipulators and the Product of Exponentials

Concerning the general equation for the forward kinematic mapping from Equation (2.43), we can now easily derive the *Product-of-Exponentials* (PoE) formula, as it was first published in [Brockett 1984][13]

$$T(\boldsymbol{\theta}) = \prod_{i=1}^{j} T(\theta_i) = \prod_{i=1}^{j} \exp\left(\theta_i G_{\boldsymbol{s_i}}\right). \tag{2.57}$$

This is simply a sequence of transformations as derived for a revolute joint in Equation (2.55). A caveat is that the $G_{\boldsymbol{s_i}}$ are always computed with respect to $T(\theta_{i-1})$, i.e., it is not possible to obtain $G_{\boldsymbol{s_i}}$ directly, if we do not know the pose of the joint $i-1$.

The above Equation (2.57) only applies to arbitrary $\boldsymbol{\theta'}$, whenever the $G_{\boldsymbol{s_j}}$ are computed according to a current configuration, i.e., the current joint angles $\boldsymbol{\theta}$. In general though, this is by no means a limitation, as this "home" configuration can be taken to be whatever suitable for an application. The PoE must always refer to this basis configuration

$$T(\boldsymbol{\theta})' = \left[\prod_{i=1}^{j} \exp\left(\Delta\theta_i G_{\boldsymbol{s_i}}\right)\right] T(\boldsymbol{\theta}) \tag{2.58}$$

where $\Delta\theta = \theta'_i - \theta_i$ and the joint screws $G_{\boldsymbol{s_i}}$ are computed with respect to $T(\boldsymbol{\theta})$. In other words, the product of exponentials only acts as an increment $T(\boldsymbol{\Delta\theta})$ on a current configuration $T(\boldsymbol{\theta})$. For a serial manipulator with j revolute joints this

[13] A little remark on the notation within this section: to avoid confusion, indices of joints are written in subscript, indices of time are written in superscript.

results in

$$\begin{aligned} T(\boldsymbol{\theta})' &= T(\boldsymbol{\Delta\theta}) \cdot T(\boldsymbol{\theta}) \\ &= T_{\boldsymbol{s_1}}(\Delta\theta_1)\; T_{\boldsymbol{s_2}}(\Delta\theta_2) \cdots T_{\boldsymbol{s_j}}(\Delta\theta_j) \cdot T(\boldsymbol{\theta}) \\ &= \exp(\Delta\theta_1 G_{\boldsymbol{s_1}}) \exp(\Delta\theta_2 G_{\boldsymbol{s_2}}) \cdots \exp(\Delta\theta_j G_{\boldsymbol{s_j}}) \cdot T(\boldsymbol{\theta}). \end{aligned} \tag{2.59}$$

As an example it is shown, that of course, also following this methodology, not changing the joint angles (i.e., applying a zero increment $\boldsymbol{\Delta\theta} = \boldsymbol{\theta}' - \boldsymbol{\theta} = \mathbf{0}$) does not affect the end-effector pose. Evaluating Equation (2.58) with the infinitesimal generators G_i of $SE(3)$ and

$$\exp(\theta G_{\boldsymbol{s}_i}) = \exp\left(\theta \sum_{i=1}^{6} s_i G_i\right) \tag{2.60}$$

from Equation (2.55) evaluates to

$$T(\boldsymbol{\theta})' = T(\boldsymbol{\Delta\theta})T(\boldsymbol{\theta}) = \left[\prod_{i=1}^{j} \exp\left(0 \sum_{i=1}^{6} s_i G_i\right)\right] T(\boldsymbol{\theta}) = T_{id} \cdot T(\boldsymbol{\theta}) = T(\boldsymbol{\theta}) \tag{2.61}$$

as expected.

The Time Dimension

Now we want to consider a sequence of configurations in time starting with an arbitrary configuration $\boldsymbol{\theta}^0$. We are then chaining transformations, i.e. increments for $\boldsymbol{\Delta\theta}^t = \boldsymbol{\theta}^t - \boldsymbol{\theta}^{t-1}$

$$T(\boldsymbol{\theta}^t) = \left[\prod_{i=1}^{t} T(\boldsymbol{\Delta\theta}^i)\right] T(\boldsymbol{\theta}^0). \tag{2.62}$$

For robotic applications this is a useful result, because we do not explicitly use absolute values of the joint angles in order to compute the forward kinematics once we know the home configuration. Instead, $\boldsymbol{\Delta\theta} = \mathbf{0} \in \mathbb{R}^n$ refers to the identity transformation T_{id}, which can be altered incrementally by modifications of the joint angles θ_i. Briefly, in the following an approach for finding the joint screw $G^t_{\boldsymbol{s_i}}$ for a specific joint j_i at some point in time t is described.

From Section 2.3.2, we know that two points on the rotation axis of the joint are needed to determine the screw, the origin $\boldsymbol{x_o}$ and a point translated one unit on the axis $\boldsymbol{x_p}$. The parameters of the Lie algebra element $\boldsymbol{s} = (\boldsymbol{x}, \boldsymbol{y})$ were then given by

$$\boldsymbol{x} = \boldsymbol{x_p} - \boldsymbol{x_o} \quad \text{and} \quad \boldsymbol{y} = \boldsymbol{x_o} \times \boldsymbol{x}.$$

The corresponding screw $G_{\boldsymbol{s}}$ was hence defined as

$$G_{\boldsymbol{s}} = \begin{pmatrix} [\boldsymbol{x}]_\times & \boldsymbol{y}^T \\ 0 & 0 \end{pmatrix}.$$

In order to obtain the joint screw, we have to evaluate the above equation EQUATION (2.62) with respect to a spatial and a temporal dimension: the index of the joint j, and the point on the time-line t. In relation to these parameters we define a Lie algebra element

$$\boldsymbol{s}_j^t = (\boldsymbol{x}_j^t, \boldsymbol{y}_j^t), \tag{2.63}$$

where the axis of rotation $\boldsymbol{x}_j^t$ and its moment $\boldsymbol{y}_j^t$ can be calculated utilizing recursive application of the transform $T_j(\boldsymbol{\theta}^t)$ specific to joint j and timestep t,

$$\begin{aligned} T_j(\boldsymbol{\theta}^t) &= \left[\prod_{i=1}^{j} \exp\left(\theta_i^t - \theta_i^{t-1}\right) G_{\boldsymbol{s_i}}^{t-1}\right] T_j(\boldsymbol{\theta}^{t-1}) \\ &= \left[\prod_{k=1}^{t} \left[\prod_{i=1}^{j} \exp\left(\theta_i^k - \theta_i^{k-1}\right) G_{\boldsymbol{s_i}}^{k-1}\right]\right] T_j(\boldsymbol{\theta}^0). \end{aligned} \tag{2.64}$$

N.b. in a practical implementation, of course the recursion should be avoided by temporarily storing the transforms $T_j(\boldsymbol{\theta}^t)$ and Lie algebra elements $\boldsymbol{s_j}$ of a current timestep, before proceeding with the next modification on the joint angles $\boldsymbol{\Delta\theta}^t$.

Explicitly evaluated for a specific joint j of a serial manipulator and some timestep t, the screw parameters are finally given as

$$\boldsymbol{x_j} = T_j(\boldsymbol{\theta}^t)\begin{pmatrix}0\\0\\1\\1\end{pmatrix} - T_j(\boldsymbol{\theta}^t)\begin{pmatrix}0\\0\\0\\1\end{pmatrix} \quad \text{and} \quad \boldsymbol{y_j} = T_j(\boldsymbol{\theta}^t)\begin{pmatrix}0\\0\\0\\1\end{pmatrix} \times \boldsymbol{x_j}. \tag{2.65}$$

Jacobians the Easy Way

Considering efficient control of a robotic manipulator, it is essential to derive means for calculating joint velocities and in close relation to that, give the basic formulae for operational space velocities.

As shown in the previous section, the transformation corresponding to the end-effector pose, specified by the screws $G_{\boldsymbol{s_j}}$ of each of the joints j of the robot and based on a home configuration $\boldsymbol{\theta} = \mathbf{0}$ is given by the product-of-exponentials from above

$$T_j(\boldsymbol{\theta}) = \prod_{i=1}^{j} \exp\left(\theta_i G_{\boldsymbol{s_i}}\right).$$

Next, a derivative of the above formula with respect to time has to be calculated in order to compute the velocity of a point in operational space $\boldsymbol{p} = (x, y, z)^T$

$$\begin{pmatrix} \dot{\boldsymbol{p}} \\ 1 \end{pmatrix} = \frac{d}{dt} T(\boldsymbol{\theta}) \begin{pmatrix} \boldsymbol{p} \\ 1 \end{pmatrix}. \tag{2.66}$$

The next paragraphs describe an efficient approach to derive this differential $\frac{d}{dt}T(\boldsymbol{\theta})$. First, the partial derivative of this formula with respect to one of the angles θ_i is given by

$$\frac{\partial}{\partial \theta_i} T(\boldsymbol{\theta}) = \frac{\partial}{\partial \theta_i} \prod_{i=1}^{j} \exp\left(\theta_i G_{\boldsymbol{s_i}}\right) = G_{\boldsymbol{s_i}} T(\boldsymbol{\theta}) \tag{2.67}$$

because joint screws $G_{\boldsymbol{s_j}}$ with $j \neq i$ vanish in the derivative. As a joint angle θ_i is in general subject to time, i.e. defined by the angular velocity, we can then specify this partial derivative with respect to t to be

$$\frac{\partial}{\partial \theta_i} \frac{d\theta_i}{dt} T(\boldsymbol{\theta}) = G_{\boldsymbol{s_i}} T(\boldsymbol{\theta}) \cdot \frac{d}{dt}\theta_i = G_{\boldsymbol{s_i}} T(\boldsymbol{\theta}) \cdot \dot{\theta}_i, \tag{2.68}$$

and finally the time derivative of the product of exponentials formula, considering the home position $\boldsymbol{\theta}^0 = \mathbf{0} = (0, \ldots, 0)^T$ and partial derivatives for all joints, can be written as a sum

$$\begin{aligned} \frac{d}{dt} T(\boldsymbol{\theta})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^0} &= \sum_{i=1}^{j} \left(G_{\boldsymbol{s_i}} T(\boldsymbol{\theta}) \cdot \dot{\theta}_i \right) \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^0} \\ &= \sum_{i=1}^{j} \left(G_{\boldsymbol{s_i}} I_4 \dot{\theta}_i \right) \\ &= \dot{\theta}_1 G_{\boldsymbol{s_1}} + \dot{\theta}_2 G_{\boldsymbol{s_2}} + \cdots + \dot{\theta}_j G_{\boldsymbol{s_j}}. \end{aligned} \tag{2.69}$$

Generalizing this result to any position to become the home position only requires to change the $G_{\boldsymbol{s_i}}$ to reflect the chosen home position's joint screws at some point in time t, as discussed in the last section. Hence, in order to indicate this arbitration, again the time index t has to be added: $G_{\boldsymbol{s_i}} \rightarrow G^t_{\boldsymbol{s_i}}$

$$\frac{d}{dt} T(\boldsymbol{\theta}) = \dot{\theta}_1 G^t_{\boldsymbol{s_1}} + \dot{\theta}_2 G^t_{\boldsymbol{s_2}} + \cdots + \dot{\theta}_j G^t_{\boldsymbol{s_j}} \tag{2.70}$$

This formula is rather simple, and in particular useful, as it provides the means for relating changes of the joint angles $\boldsymbol{\theta}$ with respect to time, i.e., joint angular

velocities to displacement (translational) velocities of the end-effector, or rather any point in the reference frame of the end-effector[14], given by a position $\boldsymbol{p} = (x, y, z)^T$

$$\begin{pmatrix} \dot{\boldsymbol{p}} \\ 1 \end{pmatrix} = \left(\dot{\theta}_1 G^t_{\boldsymbol{s_1}} + \dot{\theta}_2 G^t_{\boldsymbol{s_2}} + \cdots + \dot{\theta}_j G^t_{\boldsymbol{s_j}} \right) \begin{pmatrix} \boldsymbol{p} \\ 1 \end{pmatrix} \tag{2.71}$$

However, the above result is only one side of the coin. We can also use this result to not only specify a change of the position of a rigid body in space, but moreover to describe the change of orientation simultaneously. This leads to a complete operational space dynamics formalism.

We have already stated before, that in general a pose needs to be described by at least six parameters, three for position and three for orientation. Moreover, we have already defined formalisms to describe these parameters efficiently. The following uses the matrix representation, i.e., by viewing a pose as a transformation $T \in SE(3)$ defined by a 4×4 matrix

$$T = \begin{pmatrix} \boldsymbol{R} & \boldsymbol{t} \\ 0 & 1 \end{pmatrix}$$

and a representation for changes on that transformation by rotations θ around the z-axis of any revolute joint

$$T' = \Delta T \cdot T \quad \text{with} \quad \Delta T = \exp\left(\theta G_{\boldsymbol{s}}\right),$$

where the Lie algebra element $\boldsymbol{s} = (\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^6$ unfolds into an angular velocity part $\boldsymbol{x}$ of the joint rotation and a linear velocity part $\boldsymbol{y}$ such that the screw $G_{\boldsymbol{s}}$ is specified

$$G_{\boldsymbol{s}} = \begin{pmatrix} [\boldsymbol{x}]_\times & \boldsymbol{y}^T \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -x_3 & x_2 & y_1 \\ x_3 & 0 & -x_1 & y_2 \\ -x_2 & x_1 & 0 & y_3 \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{2.72}$$

These definitions can now be used in order to describe the *change* of a transformation with respect to time. This can be achieved by giving a formula that relates the transformation screw parameters, the axis of a rotation $\boldsymbol{x}$ and the moment $\boldsymbol{y}$, to t. The result is called a *velocity screw*. Essentially, we can find the parameters $\boldsymbol{s}^k(t)$ of the velocity screw describing the end-effector, i.e., a Lie algebra element to the screw motion subject to t, by linear combination of the joint screws.

Considering a robot with j revolute joints in a sequence, the joint screws were then described by Lie algebra elements $\boldsymbol{s}^k_i$ at some home configuration $\boldsymbol{\theta}^k$. This home

[14] Lest to forget it in deed also holds for any of the joint reference frames, not only for the end-effector.

configuration is arbitrary, so it won't be needed below. In EQUATION (2.43) the joint screws were constructed to be subject to a single "scale" parameter $\Delta\theta$ each, an increment on the rotation angle of the joint. With respect to time, these scales then become $\dot{\boldsymbol{\theta}} = \boldsymbol{\Delta\theta}/t$ (angular velocities) and the velocity screw of the end-effector can be specified component-wise

$$\boldsymbol{s}^k(t) = (\boldsymbol{x}, \boldsymbol{y}) = \left(\dot{\theta}_1 \boldsymbol{s}_1^k + \dot{\theta}_2 \boldsymbol{s}_2^k + \cdots + \dot{\theta}_j \boldsymbol{s}_j^k\right). \tag{2.73}$$

One can easily see, that the corresponding screw $G_{\boldsymbol{s}^k}$ satisfies

$$G_{\boldsymbol{s}^k} = \frac{d}{dt} T(\boldsymbol{\theta}^k) = \dot{\theta}_1 G^k_{\boldsymbol{s_1}} + \dot{\theta}_2 G^k_{\boldsymbol{s_2}} + \cdots + \dot{\theta}_j G^k_{\boldsymbol{s_j}}, \tag{2.74}$$

so EQUATION (2.73) is just a different method (and notation) for expressing EQUATION (2.70) at some configuration $t = k$, but it has an advantage: in particular, for the case of a robot with six sequential revolute joints, we can write the mapping $f : \mathbb{R}^6 \to \mathbb{R}^6$ relating the angular velocities of the joints with the operational space velocities of the end-effector defined by the velocity screw in a more compact way. By factoring out $\dot{\boldsymbol{\theta}}$ in the transpose of EQUATION (2.73) for six joints the result can be written as

$$\boldsymbol{s}^k(t)^T = \begin{pmatrix} \boldsymbol{x}^T \\ \boldsymbol{y}^T \end{pmatrix} = \left(\boldsymbol{s}_1^{kT} \;\middle|\; \boldsymbol{s}_2^{kT} \;\middle|\; \boldsymbol{s}_3^{kT} \;\middle|\; \boldsymbol{s}_4^{kT} \;\middle|\; \boldsymbol{s}_5^{kT} \;\middle|\; \boldsymbol{s}_6^{kT}\right) \dot{\boldsymbol{\theta}}^T. \tag{2.75}$$

This compilation of the joint screws $\boldsymbol{s}_i$ into a quadratic 6×6 matrix defines a matrix of partial derivatives with respect to the angular velocities

$$J^k = \left(\boldsymbol{s}_1^{kT} \;\middle|\; \boldsymbol{s}_2^{kT} \;\middle|\; \boldsymbol{s}_3^{kT} \;\middle|\; \boldsymbol{s}_4^{kT} \;\middle|\; \boldsymbol{s}_5^{kT} \;\middle|\; \boldsymbol{s}_6^{kT}\right). \tag{2.76}$$

This matrix is commonly called the *Jacobian*[15] matrix J^k for some current configuration $\boldsymbol{\theta}^k$ at some timestep $t = k$. This result expresses how changing the joint angles $\boldsymbol{\theta}$ of the robot at some angular velocity $\dot{\boldsymbol{\theta}}$ affects the pose of the robot's end-effector with an operational space velocity ($\boldsymbol{\omega}$ refers to angular, $\boldsymbol{v}$ to linear velocity)

$$\dot{\boldsymbol{p}} = (\boldsymbol{\omega}, \boldsymbol{v})^T = J^k \dot{\boldsymbol{\theta}}. \tag{2.77}$$

For small timesteps $\Delta t \to 0$, the velocity screw for the end-effector in EQUATION (2.77) can even be used directly to approximate the relation of angle increments $\boldsymbol{\Delta\theta} = \dot{\boldsymbol{\theta}} \cdot \Delta t$ to operational space increments

$$\boldsymbol{\Delta p} = \dot{\boldsymbol{p}} \cdot \Delta t = (\boldsymbol{\omega}\Delta t, \boldsymbol{v}\Delta t)^T, \tag{2.78}$$

[15] This notation of the Jacobian is based on [SELIG 2005, P. 72].

so

$$\boldsymbol{\Delta p} = J^k \boldsymbol{\Delta \theta}. \tag{2.79}$$

Finally, using the Lie group element corresponding to that increment, which is actually an element of $se(3)$, the next end-effector pose $T(\boldsymbol{\theta}^{k+1})$ after $\Delta t = t^{k+1} - t^k$ would be

$$\begin{aligned} T(\boldsymbol{\theta}^{k+1}) &= T(\boldsymbol{\Delta\theta}) T(\boldsymbol{\theta}^k) \\ &= \exp(G_{\boldsymbol{\Delta p}}) T(\boldsymbol{\theta}^k) \\ &= \exp \begin{pmatrix} [\boldsymbol{\omega}\Delta t]_\times & \boldsymbol{v}\Delta t \\ 0 & 0 \end{pmatrix} T(\boldsymbol{\theta}^k). \end{aligned} \tag{2.80}$$

Of course, Equation (2.79) is only exact for infinitesimally small timesteps, i.e. the limit $\Delta t \to 0$, otherwise it is only an approximation. This corresponds to the fact, that the Lie algebra $se(3)$ is only a tangent vector space on the manifold of transformations in $SE(3)$ at the identity. Still, for small $\boldsymbol{\Delta\theta}$, i.e., small angle increments, the solutions are sufficiently accurate, as will be discussed later.

In the next chapter the presented approach for modeling manipulator kinematics on Lie algebras is extended towards more general sensorimotor applications. Thus, the algebra elements are embedded into a descriptor and can hence be used for modeling processes in the perception domain on the one hand and for representing association of sensor and coordination with actuator data (and vice versa) on the other hand.

Chapter 3

Activation and the Lie Descriptor

Contents

THE key-idea in this thesis can be summarized in the following rationale: in an artificial agent, i.e. a robotic system, sensorimotor activation can be represented using Lie algebraic methods and this representaion unifies and simplifies computation in the action, perception, and coordination domains on a sensorimotor level. The unification takes place in the activation association layer (FIGURE 3.1), where sensor stimuli, action command feedback, and cognitive processes are brought together and from which activation is propagated.

In this argument, first, a clarification on the term *activation* is presented in SECTION 3.1. Then in SECTION 3.2, Lie algebras are discussed as a means for representing activation in a sensorimotor context for an artificial agent. Necessary mathematics for mapping from the environment context into the descriptor space and back are shown in SECTION 3.3.

3.1 Activation

While in biological systems the term activation is generally used with respect to chemo-electrical activity in the neural structures of an agent (electrical activation of receptors and release of chemicals, the neurotransmitters), in the robotic domain no equivalent to this definition per se exists. Computer scientists developed a notion of activation which is used for example in artifical neural networks (ANNs). There activation is usually represented as a numbered value in one or many neural units. Whereas this concept is general and does not refer to a specific domain or task, nor to a dedicated hierachical level in the neural processing architecture, the concept of activation for sensorimotor systems presented in the following does actually relate to a domain: the world system, i.e. the task evironment of a situated agent. In this way it follows the ideas of embodiment theory introduced earlier, claiming that a

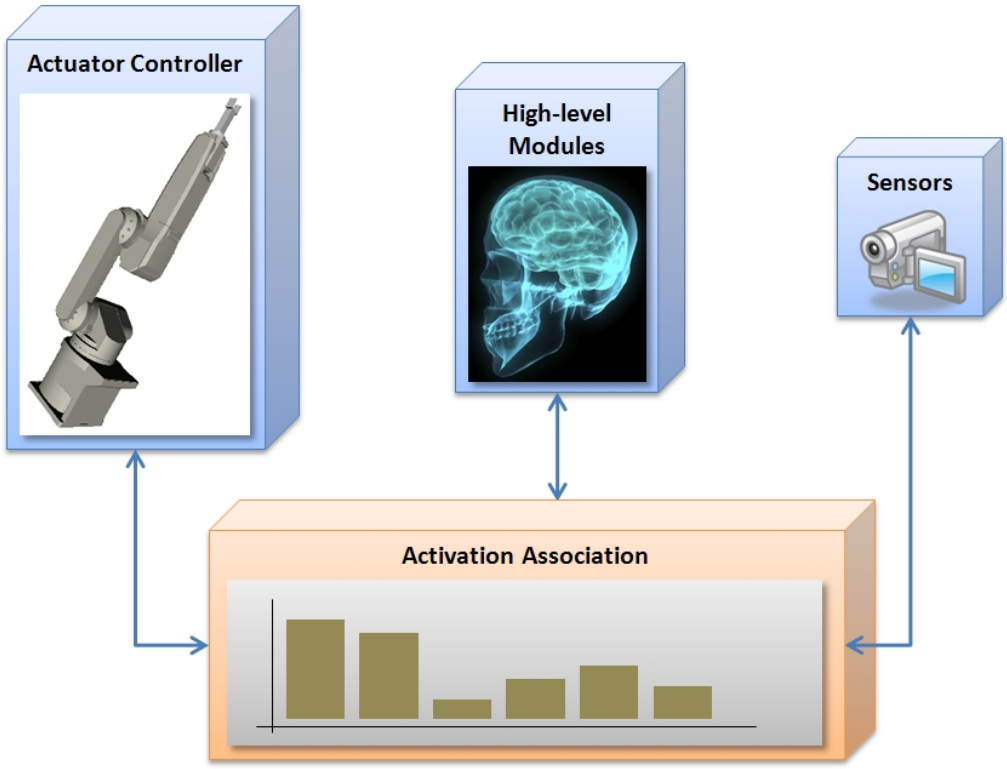

Figure 3.1 *In the activation association layer Lie algebraic features are combined and propagated.*

stimulus (i.e. activation) for sensorimotor coordination is in general a projection of direct interaction with the environment.

Consequently, the level of abstraction an activation on a sensorimotor level refers to is relatively low. It is in fact directly connected to the sensorimotor coordination domain and thus includes means for referencing to ("grounding" in) the task environment. The structure introduced in the next section resembles this finding, while it implies no restriction on more abstract cognitive processing in higher-level processes. Thus, it provides for a basis to evaluate the "what" questions on sensory stimuli in higher levels of abstraction and alongside does not loose the "where" reference of a stimulus[1]. The other way round the activation structure presented here has means to enrich abstract activation with a specific spatial context. This is for example necessary to generate motion of an actuator through motor commands.

Finally, an activation always appears in a temporal context. While in biological

[1] Embodied conceptual knowledge theory refers to the "where" reference as a parametrization of a concept, where the concept itself refers to the "what" information [Gallese and Lakoff 2005]. This is given for reference and shall not be a matter of discussion in this work.

systems this aspect is characterized through natural delays on neurotransmission, in an artificial agent, an activation structure as introduced in this work needs to store its temporal reference explicitly.

In the proposed software architecture activation evaluation is encapsulated from the low-level hardware devices (manipulator, sensors) of the robotic system used in this work as shown in FIGURE 3.1. Therefore, the central intersection instance of activation potentials from all modules, i.e. the *activation association* layer, is also the module in the robot control architecture, where coordination of behavior and processing of perceptive input data are unified.

3.2 Lie Algebraic Activation Features

In this section Lie groups and in particular Lie algebras are introduced for a specific task: representation of activation in sensorimotor systems. The general goal of this attempt is to integrate sensor and motor effects, as well as coordinative processes into a common activation reference space. Therefore, a novel activation descriptor is derived for applications discussed in the following chapter. The most relevant group in this scenario is $SE(3)$ and its algebra $se(3)$, i.e. the special euclidian group of three dimensions.

As stated before, in general a descriptor for activation always belongs to a spatial and a temporal context. Spatial, as it corresponds to a certain pose in the world; and temporal, as it occurs at a certain time. In order to use activation descriptors for attention purposes or sensorimotor association the context information has to be represented programmatically in an adequate data structure.

The time context can be specified with a 2D vector $\boldsymbol{t}$ referring to a time interval (timestamp and duration, or two timestamps – beginning and end) to which the activation applies. The spatial context can be represented in an absolute transformation reference system. A concrete pose $\boldsymbol{p}$ to which a descriptor applies can then be parametrized according to the degrees of freedom in the reference space. In APPENDIX A useful reference space representations for robotics are discussed in greater detail – for example in $SE(3)$ it usually is sufficient to use a 6D motion screw representation or a decoupled representation comprising a displacement vector and a rotation quaternion.

In the following an activation descriptor $\boldsymbol{s}$, attributed with a spatial reference $\boldsymbol{p}$ and temporal context $\boldsymbol{t}$, will be referred to as an *activation feature*. Hence the triple

uniquely identifying a feature can be written as

$$F = \{\boldsymbol{s}, \boldsymbol{p}, \boldsymbol{t}\}\,, \tag{3.1}$$

and allows for an efficient mapping between the activation space and the origin of the feature. The spatio-temporal context $\boldsymbol{p}$ and $\boldsymbol{t}$ will occasionally also be called *uplink* of a feature in the following, as it links from the differential Lie algebra (and an element of this algebra $\boldsymbol{s}$) to the absolute reference space. Compared to the direct representation of activation within the sensor or actuator reference space (as for example the 2D intensity image representation from earlier work [Müller and Knoll 2009a]) this has significant advantages:

1. It is easy to take the temporal context into account, as it is abstracted from the n-dimensional sensor or actuator space with a consistent 2D vector, as opposed to representation of time context for all concurrent or non-concurrent features in the same n-dimensional frame.
2. Features do not have to be represented in an absolute spatial structure (e.g., the 2D plane), but are instead only listed in a sequential one-dimensional buffer. Many efficient data structures exist for such buffers which allow for useful operations on the data, for examples for sorting by relevance within priority queues or ring-buffers, to name but a few.

Apart from the spatio-temporal reference $\boldsymbol{p}$ and $\boldsymbol{t}$ above, it has been implicitly stated, what actually makes the activation in an activation descriptor: the Lie algebra element $\boldsymbol{s}$ corresponding to a transformation increment $T(\boldsymbol{s})$ on a pose $\boldsymbol{p} \equiv T_{\boldsymbol{p}}$, such that

$$T_{\boldsymbol{p}'} = T(\boldsymbol{s}) \cdot T_{\boldsymbol{p}}. \tag{3.2}$$

With respect to the introduction on exponential mapping in Section 2.2.4, in particular a zero-valued activation $\boldsymbol{s} = \boldsymbol{0}$, according to Equation (2.42), gives

$$T(\boldsymbol{s}) = \prod_{i=1}^{n} \exp(s_i G_i) = \prod_{i=1}^{n} \exp \begin{pmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{pmatrix} = T_{id}. \tag{3.3}$$

For sensorimotor processes this mathematical relationship can be interpreted as follows: no activation in the system exactly refers to a zero transformation, i.e. no change in the task environment. A very useful property for applications in the robotic domain.

The next sections present some more useful properties of the activation features introduced above. From these properties methods are derived for combining features,

averaging features, and finding the difference and a distance measure between two features. The latter, i.e. evaluating feature correspondence, is probably the most relevant for sensorimotor coordination or reflexive mutual excitation, while the former are important for example for attention mechanisms, visual servoing or active perception.

3.2.1 Spatio-Temporal Correspondence

This section briefly presents an algorithm for finding spatio-temporally corresponding activation features. As stated above activation features have three components: the activation or Lie algebra element $\boldsymbol{s}$, the spatial reference $\boldsymbol{p}$, and the temporal context $\boldsymbol{t}$. Also, it has been mentioned before, that in order to find a activation correspondence between two features, they have to be represented in a common space of reference. An algorithm for finding spatio-temporal overlaps hence must comprise the following steps.

1. Represent a stimulus as an activation feature in its local frame of reference, i.e. in the perceptive field (or fields for multiple sensors) or action space for the motor domain.
2. According to calibration information, project the activation features into a common space of reference. In this step adding redundancy when reducing the dimensionionality or adding uncertainty by extending the dimension of the reference space may have to be treated carefully.
3. Find the spatio-temporal overlap, i.e. compare the timestamps in $\boldsymbol{t}$ and evaluate the spatial contexts in $\boldsymbol{p}$, i.e. the pose distance (angular and linear differences).

In general the order of evaluating the correspondence conditions is irrelevant. However, the correspondence is subject to two criteria: one for the size of the accepted time interval overlap, and one for the accepted pose distance (also called the spatio-temporal patch below). Thus, whether two activation features are considered corresponding depends largely on the choice of these parameters. This choice clearly is subject to the task to be accomplished.

3.2.2 Combining Features

Once two features are considered belonging to the same spatio-temporal context, one can work on the activation the feature carries. This activation is represented in the Lie algebra of the common space, in particular for robotic applications, the

common space usually is $SE(3)$ and hence the Lie algebra is $se(3)$.

Unfortunately, in general, the law of exponentiation from EQUATION (2.37) does not hold in the Lie algebras of non-commutative groups. In particular in the group of rigid body transformations $SE(3)$ and its algebra $se(3)$, the relation for combining two algebra elements $\boldsymbol{x}$ and $\boldsymbol{y}$

$$\boldsymbol{x} \circ \boldsymbol{y} = \boldsymbol{x} + \boldsymbol{y} \tag{3.4}$$

is **not** valid, because $\exp(X)\exp(Y) \neq \exp(X+Y)$, where X and Y are matrices

$$X = \begin{pmatrix} [\boldsymbol{\omega}_{\boldsymbol{x}}]_\times & \boldsymbol{v}_{\boldsymbol{x}}{}^T \\ 0 & 0 \end{pmatrix}, \quad \text{and} \quad Y = \begin{pmatrix} [\boldsymbol{\omega}_{\boldsymbol{y}}]_\times & \boldsymbol{v}_{\boldsymbol{y}}{}^T \\ 0 & 0 \end{pmatrix} \tag{3.5}$$

for elements $\boldsymbol{x} \in se(3)$ as vectors $\boldsymbol{x} = (\boldsymbol{\omega}, \boldsymbol{v}) = (\omega_1, \omega_2, \omega_3, v_1, v_2, v_3)$ and $\boldsymbol{y}$ respectively. This poses the question on how two activation features can be combined correctly. To elaborate on this topic a short excursion on group commutators and Lie brackets is adequate here.

In group theory, the commutator "gives an indication of the extent to which a certain binary operation fails to be commutative"[2]. In other words, it is a measure evaluating to zero, whenever the operation **is** commutative. Formally for a commutative operation one can write

$$\boldsymbol{x} \circ \boldsymbol{y} = \boldsymbol{y} \circ \boldsymbol{x}, \tag{3.6}$$

or in matrix notation

$$XY = YX \tag{3.7}$$

respectively. Then the commutator for $se(3)$ is written as a Lie bracket in matrix notation

$$[X, Y] = XY - YX. \tag{3.8}$$

In vector notation, after multiplying out this expression, the Lie bracket for $se(3)$, i.e. the commutator of the algebra on $SE(3)$ can be written

$$[\boldsymbol{x}, \boldsymbol{y}] = [(\boldsymbol{w}_{\boldsymbol{x}}, \boldsymbol{v}_{\boldsymbol{x}}), (\boldsymbol{w}_{\boldsymbol{y}}, \boldsymbol{v}_{\boldsymbol{y}})] = \begin{pmatrix} w_{\boldsymbol{y},2}w_{\boldsymbol{x},3} - w_{\boldsymbol{x},2}w_{\boldsymbol{y},3} \\ w_{\boldsymbol{x},1}w_{\boldsymbol{y},3} - w_{\boldsymbol{y},1}w_{\boldsymbol{x},3} \\ w_{\boldsymbol{y},1}w_{\boldsymbol{x},2} - w_{\boldsymbol{x},1}w_{\boldsymbol{y},2} \\ v_{\boldsymbol{x},2}w_{\boldsymbol{y},1} - v_{\boldsymbol{y},2}w_{\boldsymbol{x},1} - v_{\boldsymbol{x},3}w_{\boldsymbol{y},2} + v_{\boldsymbol{y},3}w_{\boldsymbol{x},2} \\ v_{\boldsymbol{y},1}w_{\boldsymbol{x},1} - v_{\boldsymbol{x},1}w_{\boldsymbol{y},1} + v_{\boldsymbol{x},3}w_{\boldsymbol{y},3} - v_{\boldsymbol{y},3}w_{\boldsymbol{x},3} \\ v_{\boldsymbol{x},1}w_{\boldsymbol{y},2} - v_{\boldsymbol{y},1}w_{\boldsymbol{x},2} - v_{\boldsymbol{x},2}w_{\boldsymbol{y},3} + v_{\boldsymbol{y},2}w_{\boldsymbol{x},3} \end{pmatrix}^T, \tag{3.9}$$

[2] From Wikipedia's article on the topic (`http://en.wikipedia.org/wiki/Commutator`).

or simplifying the first three elements

$$[\boldsymbol{x},\boldsymbol{y}] = \begin{pmatrix} \boldsymbol{w}_{\boldsymbol{x}} \times \boldsymbol{w}_{\boldsymbol{y}} \\ v_{\boldsymbol{x},2}w_{\boldsymbol{y},1} - v_{\boldsymbol{y},2}w_{\boldsymbol{x},1} - v_{\boldsymbol{x},3}w_{\boldsymbol{y},2} + v_{\boldsymbol{y},3}w_{\boldsymbol{x},2} \\ v_{\boldsymbol{y},1}w_{\boldsymbol{x},1} - v_{\boldsymbol{x},1}w_{\boldsymbol{y},1} + v_{\boldsymbol{x},3}w_{\boldsymbol{y},3} - v_{\boldsymbol{y},3}w_{\boldsymbol{x},3} \\ v_{\boldsymbol{x},1}w_{\boldsymbol{y},2} - v_{\boldsymbol{y},1}w_{\boldsymbol{x},2} - v_{\boldsymbol{x},2}w_{\boldsymbol{y},3} + v_{\boldsymbol{y},2}w_{\boldsymbol{x},3} \end{pmatrix}^T . \tag{3.10}$$

Based on this notion, the group operation commutes if the Lie bracket $[\boldsymbol{x},\boldsymbol{y}] = 0$ in the corresponding algebra, i.e. a group is

$$\left.\begin{array}{rrll} \text{non-commutative,} & \text{if} & \exists \boldsymbol{x},\boldsymbol{y}: & \boldsymbol{0} \neq \\ \text{commutative,} & \text{if} & \forall \boldsymbol{x},\boldsymbol{y}: & \boldsymbol{0} = \end{array}\right\} [\boldsymbol{x},\boldsymbol{y}]\,. \tag{3.11}$$

Clearly, elements of $SE(3)$ do only commute if the rotational parts $\boldsymbol{\omega}_{\boldsymbol{x}}$ and $\boldsymbol{\omega}_{\boldsymbol{y}}$ are zero vectors, as the group comprises a semidirect product

$$SE(3) = \mathbb{R}^3 \ltimes SO(3), \tag{3.12}$$

where the special orthogonal group $SO(3)$, the group of rotations in three dimensions, is obviously non-commutative.

To answer the question of combining two elements of the Lie algebra $se(3)$ correctly, one has to map back from the algebra to the group and then map the result forward into the algebra again. Two elements $\boldsymbol{x}$ and $\boldsymbol{y}$ of the Lie algebra $se(3)$ can then be combined formally into $\boldsymbol{z}$ with

$$\boldsymbol{z} = \boldsymbol{x} \circ \boldsymbol{y} = \log(\exp(\boldsymbol{x})\exp(\boldsymbol{y})). \tag{3.13}$$

The solution to this problem is known as the *Baker-Campbell-Hausdorff* (BCH) formula. It was suggested by these three in the late 19th and early 20th century, while a general, explicit combinatoric solution was given in 1947 by [Dynkin 1947]. With the BCH one writes

$$\boldsymbol{z} = \boldsymbol{x} \circ \boldsymbol{y} = BCH(\boldsymbol{x},\boldsymbol{y}). \tag{3.14}$$

A solution can be derived from Equation (3.13) by combining the infinite logarithmic and exponential series [Varadarajan 1984]. Then, the first view terms of the likewise infinite BCH-series are given as

$$BCH(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{x} + \boldsymbol{y} + \frac{1}{2}[\boldsymbol{x},\boldsymbol{y}] + \frac{1}{12}[\boldsymbol{x}-\boldsymbol{y},[\boldsymbol{x},\boldsymbol{y}]] + \mathcal{O}(|(\boldsymbol{x},\boldsymbol{y})|^4). \tag{3.15}$$

One can see, that in case the operation commutes (as for example for translations in $SE(3)$), the Lie bracket evaluates to $[\boldsymbol{x}, \boldsymbol{y}] = 0$ and the BCH-series simplifies to

$$\exp(\boldsymbol{z}) = \exp(\boldsymbol{x}) \exp(\boldsymbol{y}) = \exp(BCH(\boldsymbol{x}, \boldsymbol{y})) = \exp(\boldsymbol{x} + \boldsymbol{y}). \tag{3.16}$$

Nevertheless, even for $se(3)$ where in general $[\boldsymbol{x}, \boldsymbol{y}] \neq 0$, the inifinite series in Equation (3.15) is not of great relevance, as the logarithmic and the exponential mappings have a closed form in the context of rigid body transformations and thus the series needs not to be evaluated thoroughly. The closed forms will be elaborated on in Section 3.3.1 and Section 3.3.2 below, so at this point, it is only stated, that Equation (3.14) can be solved efficiently in this domain.

Finally, a correctly combined feature $F_{xy} = F_x \circ F_y$ can thus be written as

$$F_{xy} = \{BCH(\boldsymbol{s}_x, \boldsymbol{s}_y), \bar{\boldsymbol{p}}, \bar{\boldsymbol{t}}\}, \tag{3.17}$$

where $\bar{\boldsymbol{p}}$ and $\bar{\boldsymbol{t}}$ are averaged spatial and temporal contexts respectively.

3.2.3 Feature Distance

One can now imagine a method for measuring the distance between two activations within activation features. Informally, each feature refers to two actions on the identity element in the reference space, i.e. a rotation followed by a translation.

In a first attempt, one might think that the norm of the component-wise difference of the transformation matrices corresponding to the feature activations makes a suitable measure of distance. Unfortunately, this is not the case, as the result may not be a member of the transformation group. This is due to the fact, that the group manifold is not equivalent to its tangential vector space [Govindu 2003].

Operating in the Lie algebra of the group instead gives a better solution for a distance measure. This distance is commonly called the *Riemannian* distance. The benefit is clear: applying the logarithm to a combination of two group elements **always** results in an element of the group by definition. The idea is the following. Consider the "difference" between two transformations T_x and T_y from $SE(3)$,

$$\delta T = T_y \cdot T_x^{-1}, \tag{3.18}$$

which is defined such that it transforms "forward" by T_y and then "backward" by T_x. Then δT clearly is an element of $SE(3)$ while if T_x and T_y are equal, $\delta T = T_{id}$. The *Frobenius* norm of the Lie algebra element corresponding to δT, $||\log(\delta T)||$ can

then be used as a distance measure between two features. The *Riemannian* distance is thus

$$d(T_x, T_y) = ||\log(\delta T)|| = ||\log(T_y \cdot T_x^{-1})||. \tag{3.19}$$

Using the Lie algebra elements of T_x and T_y, $\boldsymbol{x}$ and $\boldsymbol{y}$, and the BCH formula from above, this can be written as

$$d(T_x, T_y) = ||BCH(\boldsymbol{y}, -\boldsymbol{x})||. \tag{3.20}$$

Sometimes (e.g. in [Fletcher et al. 2003, Govindu 2003]) it is claimed, that it is sufficiently accurate to approximate the Riemannian distance using the $BCH(\boldsymbol{y}, -\boldsymbol{x})$ only up to the first order by simply writing

$$d(T_x, T_y) \approx ||\log(T_y) - \log(T_x)|| = ||\boldsymbol{y} - \boldsymbol{x}||. \tag{3.21}$$

Here instead it is argued, that due to the closed form of exponential and logrithmic mappings on $SE(3)$ and $se(3)$ respectively, in this domain the benefit in terms of computational costs for the evaluation vanishes almost completely, while the yielded solution for the distance measure remains precise.

Finally, the distance δF between two activation features F_x and F_y is subject to the three constituents of a feature, i.e. the activation in the Lie algebra of the reference space $\boldsymbol{s}$, the spatial context $\boldsymbol{p}$ and the corresponding time domain $\boldsymbol{t}$. For most tasks in sensorimotor robotics, the spatio-temporal correspondence is evaluated first, so for a distance measure of features only the activation difference is regarded

$$\delta F = d(F_x, F_y) \equiv d(T_x, T_y). \tag{3.22}$$

Nevertheless, for some aplications it is also necessary to weigh this function with a factor $w_{xy} \in [0, 1]$ reflecting the spatio-temporal correlation of the two features

$$\delta F_w = d_w(F_x, F_y) = w_{xy} \cdot ||BCH(\boldsymbol{y}, -\boldsymbol{x})||. \tag{3.23}$$

3.3 Activation Propagation

Nothing has yet been said about how an activation feature can be generated from sensory or motor stimuli, or from high-level intention projection. Neither has been discussed, how an activation finds its way back into the sensory, motor or high-level modules, where it is transformed into a manipulator action in case of the motor

domain, a perceptive stimulus in case of the sensor domain, or serves as input to coordinative processes in high-level regions. This process is threefold[3].

1. Evaluate a sensor-, motor-, or high-level stimulus and create a domain-specific activation from the corresponding action in the local space. Methodology is introduced to generate representations of stimuli in the visual perception domain in the corresponding Lie group for a sensor, and analogously for the motor domain and high-level processes.
2. Generate activation in a common reference space for association and modulation. This task involves mapping of elements in local spaces into the common reference space subject to calibration information. In this step adding redundancy when reducing the dimensionionality or adding uncertainty by extending the dimension of the reference space may have to be treated carefully, if local spaces and the common space differ in their dimensionality.
3. Propagate ("mirror") the associated activation into other local spaces to affect physical or cognitive processes. This comprises mapping back from the activation domain into the common reference space, as well as from the common reference space into the local spaces. Here, it is for example explained how motion of a manipulator is directly generated from activation in its local space.

3.3.1 Mapping into the Descriptor Space

It has been stated several times before, that activation can be represented as a Lie algebra element in a common reference space for sensorimotor robotic applications. For this purpose, the last sections introduced an activation feature and derived means for finding corresponding ones, evaluate averages, and computing combinations.

Nevertheless, it has not been discussed yet, how the features can be explicitly found in a sensorimotor system. This section thus in particular introduces the mathematical methodology to generate an activation feature from a corresponding transformation in $SE(3)$, as this is the most complex group of local spaces considered in this work. The common reference space hence also needs not be more complex than $SE(3)$. This and the next section focus on internal processes in the association layer.

[3] In the following, the terminus "local space" always refers to a Lie group in a processing module, i.e., the blue boxes considering FIGURE 3.1, while "common reference space" is the Lie group and algebra used in the activation association layer, i.e. the orange box in the figure.

In Section 2.2.4 it was explained that the mapping from a Lie group into the Lie algebra in general is a logarithmic mapping. This refers to the fact, that the Lie algebra forms a tangent vector space at the identity element of the group manifold. While the log-map may be complex and not unique for certain groups, for the group of rigid body transformations considered in $SE(3)$, the mapping is relatively simple and even has a closed form [Agrawal 2005]. This allows for efficient implementation and utilization within the activation feature structure directly.

The following paragraphs hence discuss the closed form of the logarithmic mapping $SE(3) \mapsto se(3)$, i.e. a mapping

$$T_{4\times4} \mapsto \boldsymbol{s} = (\boldsymbol{\omega}, \boldsymbol{v}) : \quad \boldsymbol{s} \equiv \begin{pmatrix} [\boldsymbol{\omega}]_\times & \boldsymbol{v} \\ 0 & 0 \end{pmatrix} = \log(T) = \log \begin{pmatrix} R_{3\times3} & \boldsymbol{t} \\ 0 & 1 \end{pmatrix}. \tag{3.24}$$

Incrementally, one can derive for pure translations $\boldsymbol{t}$, be it in $\mathbb{R}^2$ or $\mathbb{R}^3$, the logarithmic mapping to be the identity map

$$\log(\boldsymbol{t}) = \boldsymbol{t}. \tag{3.25}$$

Next for $SO(2) \mapsto so(2)$, one can derive the log map, such that 2×2 skew-symmetric matrix elements of $so(2)$ result from z-axis rotation matrices of $SO(2)$ with rotation about the angle θ. Formally this is

$$\log \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & -\cos\theta \end{pmatrix} = \begin{pmatrix} 0 & -\theta \\ \theta & 0 \end{pmatrix}. \tag{3.26}$$

The closed form log-mapping in the special orthogonal group $SO(3) \mapsto so(3)$, which relates rotations in 3D represented in 3×3 matrix notation to the Lie algebra element in $so(3)$, denoted by a vector $\boldsymbol{\omega}$, can be written as

$$\log(R_{3\times3}) = \frac{\phi}{2\sin(\phi)}(R_{3\times3} - R_{3\times3}^T) \equiv [\boldsymbol{\omega}]_\times \tag{3.27}$$

where $[\boldsymbol{\omega}]_\times$ is a 3×3 skew-symmetric matrix from a rotation vector $\boldsymbol{\omega} = (\omega_1, \omega_2, \omega_3)^T$

$$[\boldsymbol{\omega}]_\times = \begin{pmatrix} 0 & -\omega_1 & \omega_2 \\ \omega_1 & 0 & -\omega_3 \\ -\omega_2 & \omega_3 & 0 \end{pmatrix}. \tag{3.28}$$

That is, $|\boldsymbol{\omega}|$ can be interpreted as the rotation angle, whereas $\bar{\boldsymbol{\omega}} = \frac{\boldsymbol{\omega}}{|\boldsymbol{\omega}|}$ is the axis of rotation. Furthermore, in Equation (3.27) ϕ has to satisfy

$$sp(R_{3\times3}) = \sum_{i=0,1,2} r_{i,i} = 1 + 2\cos\phi, \tag{3.29}$$

so with $|\phi| < \pi$

$$\phi = \cos^{-1}\left(\frac{sp(R_{3\times3}) - 1}{2}\right). \tag{3.30}$$

Finally, in the closed form log-mapping $SE(3) \mapsto se(3)$, where a 4×4 matrix notation is used, one can write the mapping equation as

$$\log\begin{pmatrix} R_{3\times3} & \boldsymbol{t} \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \log(R_{3\times3}) & A\boldsymbol{t} \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} [\boldsymbol{\omega}]_\times & A\boldsymbol{t} \\ 0 & 0 \end{pmatrix} \tag{3.31}$$

where e.g. [Agrawal 2005] shows, that A can be explicitly written as

$$A = T_{id} - \frac{1}{2}[\boldsymbol{\omega}]_\times + \frac{2\sin|\boldsymbol{\omega}| - |\boldsymbol{\omega}|(1+\cos|\boldsymbol{\omega}|)}{2|\boldsymbol{\omega}|^2\sin|\boldsymbol{\omega}|}[\boldsymbol{\omega}]_\times^2 . \tag{3.32}$$

Reading Equation (3.31), one can find the linear activation part in $\boldsymbol{s} = (\boldsymbol{\omega}, \boldsymbol{v})$, this is

$$\boldsymbol{v} = A\boldsymbol{t}, \tag{3.33}$$

with A as defined above, while $\boldsymbol{\omega}$ can be directly extracted from $[\boldsymbol{\omega}]_\times$ componentwise.

To conclude, stimulus in a local reference space $SE(3)$, i.e. in the sensor, actuator or coordination domain, can be represented as an activation feature[4] using a Lie algebra element $\boldsymbol{s} \in se(3)$

$$F = \{\boldsymbol{s}, \boldsymbol{p}, \boldsymbol{t}\} = \{(\boldsymbol{\omega}, \boldsymbol{v}), \boldsymbol{p}, \boldsymbol{t}\}. \tag{3.34}$$

Lest to say constructing a Lie algebra element in $se(3)$ corresponding to a transformation in $SE(3)$ is not possible for any element of the group, but only close to the identity transform. This is due to the restriction on $|\phi|$ above. With the inverse mapping, i.e. the closed form of the exponential mapping, explained next it becomes clearer why this has to be in a π-interval around zero.

3.3.2 Mapping into a Reference Space

As claimed in Section 1.1, the algebraic representation of activation includes defining a common transformation space based on Lie groups and a description of activation in terms of Lie algebra elements for perceived and self-created changes in the environment. So the crux in the idea is representing activation by means of the Lie algebra element corresponding to a transformation close to the identity in the **common** transformation space (e.g., $\mathbb{R}^3$ in Figure 3.2).

[4] Note that $\boldsymbol{t}$ in Equation (3.34) now specifies the time context, not a translation in SE(3) as above.

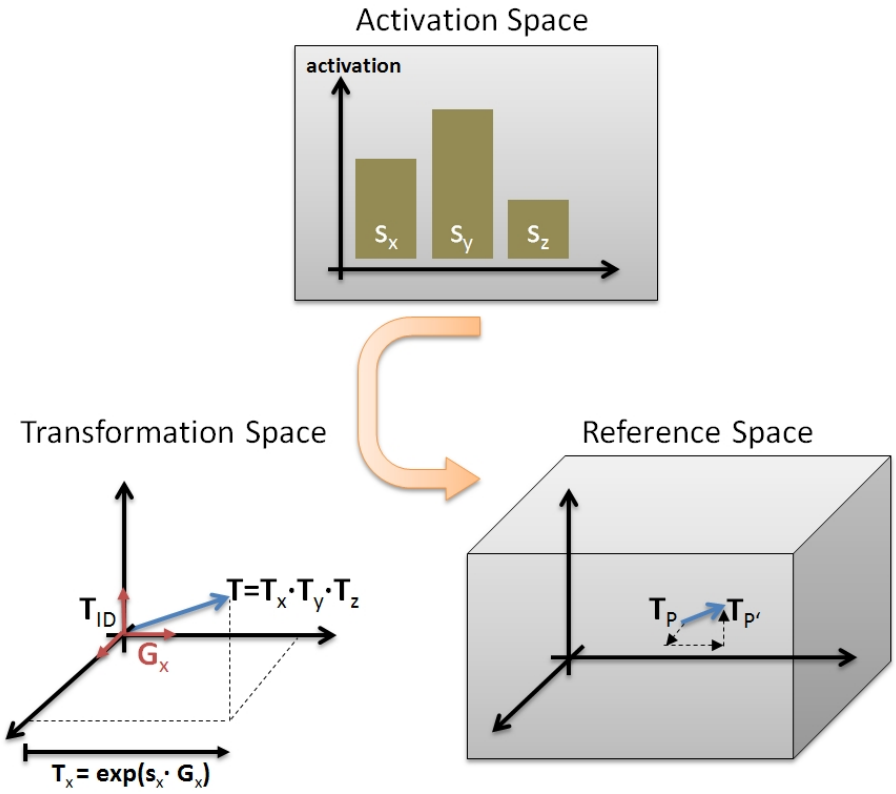

Figure 3.2 *Transformations $T_p \rightarrow T'_p$ are represented by "local offsets" from T_{id}; corresponding activation in the Lie algebra $\boldsymbol{s} = (s_x, s_y, s_z)^T$ can be transferred into a reference space with the exponential map.*

Due to the differential properties of the Lie algebra vector space, information on the origin of the activation in the world system and also the temporal context to which it applied are usually lost during projecting into the descriptor space. Thus the feature needs to carry this information in $\boldsymbol{p}$ and $\boldsymbol{t}$ as explained before.

As an activation feature is extracted from a corresponding transformation, for example a rigid body pose increment, reconstructing a transformation in a transformation group manifold after looping it through the activation association layer must be possible. This reconstruction in group theory mathematically corresponds to the exponential mapping as introduced in principle in Section 2.2.4. In the field of sensorimotor robotics, at most an exponential mapping of $SE(3) \mapsto se(3)$ is required, as this corresponds to the group of physically possible transformations (on a rigid body). The following paragraphs hence discuss this mapping $se(3) \mapsto SE(3)$. Here,

a closed form of the mapping

$$\boldsymbol{s} = (\boldsymbol{\omega}, \boldsymbol{v}) \mapsto T_{4\times4} : \quad T_{4\times4} = \exp \begin{pmatrix} [\boldsymbol{\omega}]_\times & \boldsymbol{v} \\ 0 & 0 \end{pmatrix} \tag{3.35}$$

can be derived for efficient computation in robotic scenarios. This is in fact the inverse to the logarithmic map derived in Section 3.3.1.

On pure translations, i.e. the exponential on Lie algebra elements $\boldsymbol{s} = (\boldsymbol{0}, \boldsymbol{v})$, the exponential is simply the identity map, as the inverse of an identity map is also an identity map, so

$$T(\boldsymbol{0}, \boldsymbol{v}) = \exp \begin{pmatrix} 0 & 0 & 0 & v_1 \\ 0 & 0 & 0 & v_2 \\ 0 & 0 & 0 & v_3 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3.36}$$

Next, the closed form of the exponential map $so(3) \mapsto SO(3)$ is well known and called the *Rodriguez*-formula. It is essentially a *Taylor*-expansion of the exponential series on $[\boldsymbol{\omega}]_\times$,

$$R_{3\times3} = \exp [\boldsymbol{\omega}]_\times = I + \frac{\sin|\boldsymbol{\omega}|}{|\boldsymbol{\omega}|} [\boldsymbol{\omega}]_\times + \frac{1 - \cos|\boldsymbol{\omega}|}{|\boldsymbol{\omega}|^2} [\boldsymbol{\omega}]_\times^2 . \tag{3.37}$$

The above result corresponds to a mapping from an axis-angle representation of a rotation $\boldsymbol{\omega}$, where $\bar{\boldsymbol{\omega}} = \boldsymbol{\omega}/|\boldsymbol{\omega}|$ is a unit rotation axis and $|\boldsymbol{\omega}|$ the angle of the rotation, to a 3D rotation matrix $R_{3\times3}$.

Now the exponential for $SO(3)$, the special orthogonal group of 3D rotations, and the exponential on translations in $\mathbb{R}^3$ can be utilized to formulate the closed-form exponential mapping in $SE(3)$ as

$$\exp \begin{pmatrix} [\boldsymbol{\omega}]_\times & \boldsymbol{v} \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} \exp [\boldsymbol{\omega}]_\times & \bar{A}\boldsymbol{v} \\ 0 & 1 \end{pmatrix}. \tag{3.38}$$

Here (e.g. see [Agrawal 2005]), the explicit form of the correction A on $\boldsymbol{v}$ in the above equation is given as

$$\bar{A} = I + \frac{1 - \cos|\boldsymbol{\omega}|}{|\boldsymbol{\omega}|^2} [\boldsymbol{\omega}]_\times + \frac{|\boldsymbol{\omega}| - \sin|\boldsymbol{\omega}|}{|\boldsymbol{\omega}|^3} [\boldsymbol{\omega}]_\times^2 . \tag{3.39}$$

Concluding this section, we define the projection of a descriptor into a common reference transformation space to be the exponential mapping of the Lie algebra

element in $se(3)$ to a group element of $SE(3)$. Due to the *Rodriguez*-formula this is only possible in a closed form for angular components of $\boldsymbol{\omega}$ up to $|\pi|$, because otherwise EQUATION (3.37) would not be unique and thus EQUATION (3.38) does not work.

Summary 3.1 – **Mapping To and From the Descriptor Space**

Mapping into the descriptor space requires

- Represent activation as a transformation in the local space.
- Project activation into the common reference space.
- Create the feature by applying the logarithmic mapping of $SE(3)$.

Mapping from the descriptor space includes

- Apply the exponential mapping of $se(3)$ to reconstruct the transformation for a feature.
- Project the transformation into local spaces, i.e. actuator, sensor and coordination spaces.
- Produce local representations of the transformation and induce physical activity based on the feature projection (e.g. motor commands, and cognitive or sensor activity).

Chapter 4

From Activation to Coordination

Contents

THIS chapter elaborates on how sensor, actuator and coordinative modules interact with the activation association layer in the theoretical framework. This includes methodology to transfer activation in the motor domain into manipulator control commands and to generate feedback for the activation layer in Section 4.1. In the sensor domain it is discussed, how sensori stimuli must be prepared for propagation into the association layer as activation features and how activation feedback from the layer can be used to inhibit bottom-up stimuli in order to achieve selective attentional focus (Section 4.2) before visual processing takes place.

In Section 4.3, i.e. corresponding to the coordination domain, first it is explained, how motor activation and sensor activation can be connected for sensorimotor response yielding manipulator motion or sensor stimulus mediation. Furthermore, the section shows how higher-level intentions can be forwarded into the activation layer to achieve dedicated manipulation tasks.

Each section contains application examples in real-world robotic problem domains. These problems are taken from the research scenarios detailed in Chapter 5, i.e., a human-robot interaction scenario from the JAST project and a industrial automation scenario from the SFB-T4 project.

4.1 Robot Motion From Activation

Once activity is pushed from the activation association layer into the manipulator control domain, this has to be interpreted as a robot control command there. In principle, a new Cartesian target pose is computed from the current pose and the increment corresponding to the activation and its time-context. Then a path generation sub-system generates angle configurations for the manipulator in order to reach for the desired target.

Typically several constraints have to taken into account in this process. Apart from "normal" constraints like singularity safety, first, usually the time-context given in an activation feature is much larger than the update-interval of the manipulator hardware. Second, as it is likely that new activity is introduced into the system while a motion is currently conducted, the path generation system must be realtime reconfigurable. Nevertheless the generated manipulator trajectory must comply with hardware constraints like velocity limits.

In general in the field of robotics a *path* is a line in the operational space coordinate frame of a robot, that moves the end-effector along a curve. For robotic manipulators only paths make sense, that do not require discontinuous velocity profiles. That means, the generated path should be differentiable, i.e., is sufficiently smooth with respect to time.

In the next section a method is presented to solve the problem of generating such a smooth path with all intermediate poses for a 6R manipulator in order to reach for a fixed static target pose with its end-effector (Section 4.1.1). This elegant solution for a classical problem shows the applicability of the compositional approach of group theory as it was derived in the last sections. Thereafter the compositional approach is extended further in order to be able to reach for a freely movable target without violating the smoothness constraint (Section 4.1.2). This can in fact be derived straight forward from the theoretical considerations before, i.e., the incremental pose computation.

4.1.1 Moving Towards a Static Target

The problem solved within this section is mainly a problem of continuous interpolation. The basis is the system being in a home configuration, i.e., the end-effector pose[1] is given from the forward kinematics as explained in Section 2.3.3.

Figure 4.1 shows the scenario the approach is applicable to. This scenario is also sometimes called a *point-to-point motion* trajectory, as opposed to a *path motion*, where intermediate points are specified as well [Sciavicco and Siciliano 2001]. In the scenario, the home configuration corresponds to some end-effector pose T^{home} in 3D space[2], i.e, an element of the Lie group $SE(3)$ in some representation (see Appendix A).

[1] The terms *pose* and *transformation* are interchangable with each other in this context, as a pose essentially refers to a transformation applied to the origin. In other words, the transformation specifies the act of moving while a pose specifies the state after having moved.

[2] From this section on, the indication of a transformation $T(\boldsymbol{\theta})$ being subject to a joint angle configuration $\boldsymbol{\theta}$ is occasionally omitted for readability of the equations.

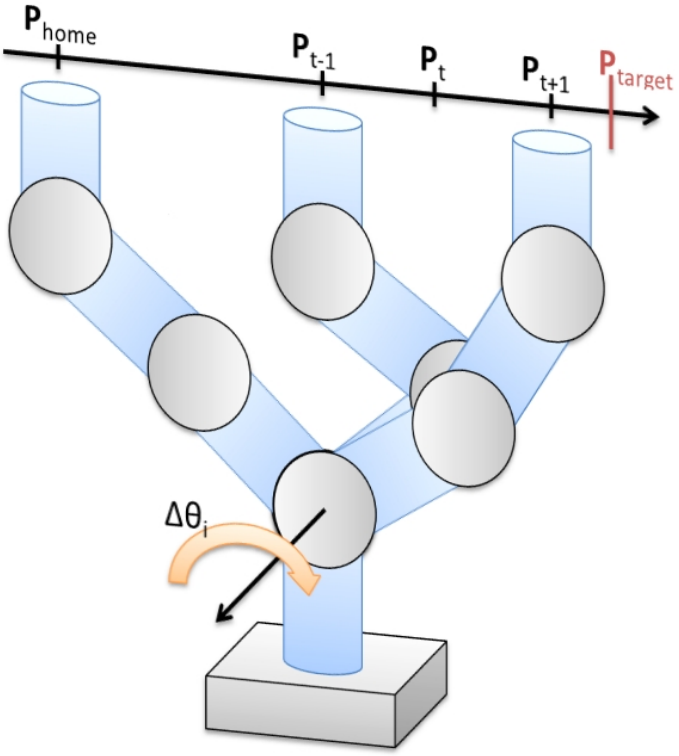

Figure 4.1 *The scenario for compositional trajectory generation with respect to a static, stationary target.*

For example, the element could be specified by its homogeneous matrix representation, a 4×4 matrix. The target pose is given by some transformation T^{target}. Intermediate poses are given relative to each other, in particular this results in a sequence

$$T^{home}, T^{home+1} \cdots T^{t-1}, T^{t}, T^{t+1} \cdots T^{target-1}, T^{target}.$$

The problem of obtaining the corresponding joint angle configuration $\boldsymbol{\theta}$ for a pose on the end-effector path T is known as the *inverse kinematics* problem. Luckily, at the end of the last section a method has been derived that enables to compute the change on the end-effector pose subject to a change of the joint angle configuration

$$\boldsymbol{\Delta p} = J^t \boldsymbol{\Delta \theta},$$

which, as stated in Equation (2.80), for $\Delta t \to 0$ delivers an arbitrary accurate approximation of

$$T(\boldsymbol{\theta}^{t+1}) = \exp(G_{\boldsymbol{\Delta p}}) T(\boldsymbol{\theta}^t). \tag{4.1}$$

Now if the current configuration $\boldsymbol{\theta}^t$ is known, as it is usually from the sequence of

angle increments on the initial values

$$\boldsymbol{\theta}^t = \boldsymbol{\theta}^0 + \sum_{k=1}^{t} \left(\boldsymbol{\Delta\theta}^k \right), \tag{4.2}$$

also the current pose is clear as explicitly derived in EQUATION (2.64) in the last section. Thus, the problem reduces to solving the inverse of EQUATION (2.79) for each timestep t

$$\left(J^t\right)^{-1} \boldsymbol{\Delta p} = \boldsymbol{\Delta\theta}. \tag{4.3}$$

This is known as the problem of inverting the *Jacobian*, which is not covered in detail in this work. Lest to say there exists a huge amount of literature on the topic though, elaborating on different methods for computing the matrix inverse (in under-, over-, and well-constrained systems) and approximations, e.g., the Jacobian transpose $J^{-1} \approx J^T$, pseudoinverse $J^{-1} \approx (J^T J)^{-1} J^T$, damped (by λ) least squares $J^{-1} \approx J^T (JJ^T + \lambda^2 I)^{-1}$ methods, and so on and so forth. Refer to e.g., [CRAIG 1955, BUSS 2009] for some basics.

As with the inverse Jacobian the mapping from a pose increment to a corresponding joint angle increment can be solved for a 6R manipulator[3], the general problem of path generation in its compositional form now is to find that increment $\Delta T = \exp(G_{\boldsymbol{\Delta p}})$ on the current end-effector pose subject to a desired velocity such that

$$T^{t+1} = \Delta T \cdot T^t. \tag{4.4}$$

A naïve approach could be the numerical evaluation of the logarithm on ΔT to find $\boldsymbol{\Delta p}$. Unfortunately, this logarithm is not unique and in general involves a lot of computation, as also the current pose would have to be inverted. The solution using this approach could be written as follows.

1. Find the scaled generator $G_{\boldsymbol{\Delta p}}$ from EQUATION (4.1) by rearrangement, so

$$G_{\boldsymbol{\Delta p}} = \log \left[(T^t)^{-1} \cdot T^{t+1} \right]. \tag{4.5}$$

2. Find the corresponding Lie algebra element

$$\boldsymbol{\Delta p} = (\boldsymbol{\omega}, \boldsymbol{v})^T \quad \text{from} \quad G_{\boldsymbol{\Delta p}} = \begin{pmatrix} [\boldsymbol{\omega}]_\times & \boldsymbol{v} \\ 0 & 0 \end{pmatrix}. \tag{4.6}$$

3. Apply the inverse Jacobian to find the joint angle increments

$$\boldsymbol{\Delta\theta} = J^{-1} \boldsymbol{\Delta p}. \tag{4.7}$$

[3] Although special cases, i.e., singular configurations, have to be taken into account separately.

A more efficient approach, as implemented in this work, uses a different method to find $\boldsymbol{\Delta p}$ directly from ΔT without explicitly computing a logarithm. Essentially it takes advantage of the fact, that the increments ΔT only occur very close to the identity T_{id}, as $\Delta t \to 0$, and that in the end-effector frame rotations are composed of rotations about the orthonormal basis vectors, i.e., the coordinate axes[4]. Thus, an analytical decomposition of the transformation corresponding to the increment in the end-effector frame can be applied [Eberly 2011].

Using such an approach for moving the end-effector of a robot towards a static target, a 3D interpolation with six degrees of freedom has to be calculated. Any discrete intermediate pose can be computed in terms of its predecessor using the recursive form as derived in Equation (4.1) in the last section. So in general, the intermediate poses can also be written as a chain in the compositional form, i.e. with respect to an arbitrary initial pose T^{home}

$$T^t = \left[\prod_{i=home+1}^{t} \exp(G_{\boldsymbol{\Delta p}^i}) \right] T^{home}. \tag{4.8}$$

Constant Velocity

Regarding the most trivial case, i.e., moving the end-effector towards a static target at a constant velocity, the problem reduces to a simple 3D *discrete equidistant* interpolation in $SE(3)$. In particular, satisfying the constant velocity constraint requires all the angular increments $\boldsymbol{r}_\Delta = \boldsymbol{\omega}\Delta t$ and linear increments $\boldsymbol{t}_\Delta = \boldsymbol{v}\Delta t$ to amount to the same absolute value

$$\forall i, j \in \{home, \ldots, target\} : \quad |\boldsymbol{r}^i_\Delta| = |\boldsymbol{r}^j_\Delta| \ \wedge \ |\boldsymbol{t}^i_\Delta| = |\boldsymbol{t}^j_\Delta|. \tag{4.9}$$

Trapezoidal or more complex velocity profiles typically result in varying absolute values of the increments $|\boldsymbol{t}_\Delta|$ and $|\boldsymbol{r}_\Delta|$ in $\boldsymbol{\Delta p}^i$ at some points or even all points. These are also discussed later on in greater detail.

In the following, solutions to the path generation problem are explained using a decoupled representation of $SE(3)$, as introduced in Section A.1,

$$T_D = \langle \boldsymbol{r}, \boldsymbol{t} \rangle.$$

The translational increments of ΔT can thus be given as the absolute difference of homogeneous coordinates of two consecutive poses. Equivalently, the translation

[4] Which in a matrix representation would result in a skew-symmetric rotation part of the transformation.

parameters of a pose given as a 4×4 matrix T^i can be obtained by retrieval of the last column, i.e.,

$$\boldsymbol{t} = \begin{pmatrix} t_x \\ t_y \\ t_z \\ 1 \end{pmatrix} = \begin{pmatrix} T_{0,3} \\ T_{1,3} \\ T_{2,3} \\ 1 \end{pmatrix}. \tag{4.10}$$

The rotation increments on the other hand are best computed using the quaternion representation and distance measures of two consecutive poses for the following reasons: unit quaternions provide an alternative representation for rotations in 3D space which, compared to Euler angles do not suffer from the *gimbal lock*, and compared to rotation matrices, are numerically more stable [Lee 1991, Vicci 2001] (a further mathematical introduction regarding mappings from $SO(3)$ to the 3-sphere S^3, $so(3)$ and $\mathbb{R}^3$, i.e., the relation to exponential mappings is discussed in Section A.1.4). The derivation of quaternions from rotation matrices, and conversions to rotation matrices and exponential rotation vectors can for example be reviewed from [Grassia 1998]. At this point, we only recall, that it requires the extension of complex numbers to a hypercomplex system with one real and three imaginary parts,

$$\boldsymbol{q} = q_0 + q_1\mathrm{i} + q_2\mathrm{j} + q_3\mathrm{k}, \tag{4.11}$$

where i, j, and k are imaginary units satisfying $\mathrm{i}^2 = \mathrm{j}^2 = \mathrm{k}^2 = \mathrm{ijk} = -1$; and q_0 is called the scalar part and the triple (q_1, q_2, q_3) is called the vector part of the quaternion.

This system is a representation of the group S^3 for the unit 3-sphere. However, for efficient computations, we utilize the scalar representation of the quaternion space $\mathbb{R}^4$ with a quaternion specified by four real-valued coefficients (q_0, q_1, q_2, q_3). Fortunately, also the mapping to $SO(3)$ only requires minimal caution and effort. Only for the reason that S^3 is a double cover of the special orthogonal group, one has to ensure the quaternions to be located on the same hemisphere of S^3 (see Section A.1.4 for details).

In general, the increments ΔT can be infinitesimally small as the transformation group $SE(3)$ is a differential manifold, i.e., a Lie group, which is a nice property when it comes to arbitrary accurate interpolation. Nevertheless, as physical robots can only receive discrete control commands at a hardware specific update interval, for robot control, the path has to be discretized with respect to that update rate u of the robot's realtime interface and the desired operational space velocity. The

pose increment thus can be written

$$\Delta T = \langle \boldsymbol{r}_\Delta, \boldsymbol{t}_\Delta \rangle \quad \text{with} \quad \langle \boldsymbol{r}_\Delta, \boldsymbol{t}_\Delta \rangle = \langle \boldsymbol{\omega} u, \boldsymbol{v} u \rangle. \tag{4.12}$$

Here, $|\boldsymbol{\omega}|$ is the angular and $|\boldsymbol{v}|$ the linear velocity, while $u = \Delta t$ corresponds to the update rate of the robot. Typically, the update rate is specified by the manufacturer, for example considering a *Mitsubishi* RV6-S robot the update rate $u = 0.00711\,\mathrm{s}$ is given. The operational space velocities are only constrained by the maximum angular velocities of the joints and amount to about $9.3\,\frac{\mathrm{m}}{\mathrm{s}}$ for displacements of the end-effector of this robot.

Note that, applying the inverse kinematics may result in joint angle velocities greater than the allowed maximum (e.g. close to a singularity[5]). In this case, the increment has to be rescaled according to the exaggerated speed at the cost of a non-constant velocity profile. In practice, here the best solution would be to keep the desired operational space velocity well below the maximum, in particular for end-effector poses close to the workspace boundaries.

As the path is to be computed in a compositional way, i.e., the next T^{t+1} only relies on the current T^t and the target T^{target}, the approach resolves to computing the distance between the two and then "scaling" it according to the constraint from Equation (4.9). This component-wise distance to the target comprises a combination from the translational distance $\boldsymbol{t}_\Delta$ and the rotational (i.e. quaternion) distance $\boldsymbol{q}_\Delta$. While the translational part can be directly computed as a difference of homogeneous coordinates

$$\boldsymbol{t}_\Delta = \boldsymbol{t}_{target} - \boldsymbol{t}_t, \tag{4.13}$$

for the rotation distance, the interpolating quaternion is computed

$$\boldsymbol{q}_\Delta = \boldsymbol{q}_{target} \boldsymbol{q}_t^{-1}. \tag{4.14}$$

The distance in quaternion space is ambiguous as each rotation in $SO(3)$ maps to antipodes in S^3 (the double cover of $SO(3)$ mentioned before). Each orientation thus has two representations in the unit 3-sphere. Therefore, to ensure to get the rotation distance components right, first, the sign of the dot product of the quaternions $\boldsymbol{q}_t \cdot \boldsymbol{q}_{target}$ is evaluated. Two quaternions are mapped to the same hemisphere in S^3 if their dot product is positive [Grassia 1998]. If this should not be the case,

[5] Singular configurations can be detected easily by checking the joint screws from which the Jacobian is generated (see Section 2.3.4), i.e., whenever the joint screws do not form a basis of $SE(3)$ the system is exposed to a singular configuration.

one of the quaternions $\boldsymbol{q}$ has to be replaced by its negation $-\boldsymbol{q}$ for all interpolation equations.

Since a distance measure from the current pose to the target pose has already been computed, now this measure is used to compute the next pose T^t from an increment ΔT and T^{t-1}. For a constant velocity profile, again composing from translational and rotational parts, we can first write the translational part as follows.

1. After u seconds, according to the desired speed $v_T = |\boldsymbol{v}| = \sqrt{v_x^2 + v_y^2 + v_z^2}$ in $\frac{\mathrm{m}}{\mathrm{s}}$, the end-effector needs to have moved $v_T \cdot u$ units into an arbitrary direction.
2. The translational fraction
$$c_T = \frac{v_T \cdot u}{|\boldsymbol{t}_\Delta|} \tag{4.15}$$
then gives the Euclidian distance the end-effector may move within one update cycle (one update step).
3. As this distance has to be distributed to the three components of the translation, we simply multiply it with the vector of distances to the target
$$\boldsymbol{t}_\delta = c_T \boldsymbol{t}_\Delta = \frac{v_T \cdot u}{|\boldsymbol{t}_\Delta|} \boldsymbol{t}_\Delta. \tag{4.16}$$

To work out the rotation increment the quaternion distances as described above are utilized.

1. After u seconds, according to $v_R = |\boldsymbol{\omega}| = \sqrt{\omega_x^2 + \omega_y^2 + \omega_z^2}$, i.e., the desired angular velocity, the end-effector needs to have rotated $v_R \cdot u$ units about an arbitrary axis.
2. The total rotation increment, i.e., the rotation required to get from the current orientation of the target orientation, $\boldsymbol{q}_\Delta$, corresponds to a rotation in axis angle representation $R(\theta, \boldsymbol{x})$, where $\boldsymbol{x}$ denotes the axis $\in \mathbb{R}^3$ and θ the angle (see Section A.1.3). This angle can be retrieved directly from the scalar part q_0 of the quaternion $\boldsymbol{q}_\Delta$ with
$$\theta_\Delta = 2 \cdot \cos^{-1}(q_0). \tag{4.17}$$
This is considered subject to the norm of the desired angular velocity v_R. Now, we can determine the fraction feasible in one update step
$$c_R = \frac{v_R \cdot u}{|\theta_\Delta|}. \tag{4.18}$$
3. Finally, the quaternion $\boldsymbol{q}_\delta$ representing the rotation increment can be obtained using Shoemake's *spherical linear interpolation* [Shoemake 1985], the

so-called SLERP

$$\boldsymbol{q}_\delta = (\boldsymbol{q}_\Delta)^{c_R}, \tag{4.19}$$

where powers of a (unit) quaternion can be computed efficiently using the *versor*-form of the quaternion $\boldsymbol{q} = \exp(\theta_\Delta \boldsymbol{x})$ with $\boldsymbol{x} = (q_1, q_2, q_3)^T$. The increment $\boldsymbol{q}_\delta$ then is

$$\boldsymbol{q}_\delta = \cos(c_R\theta_\Delta) + \sin(c_R\theta_\Delta)q_1\mathrm{i} + \sin(c_R\theta_\Delta)q_2\mathrm{j} + \sin(c_R\theta_\Delta)q_3\mathrm{k}. \tag{4.20}$$

At last, to find the increment $\Delta T = \delta T$ for the current step according to Equation (4.4) is straightforward. One only has to consider, that in order to reach both orientation and position of the target end-effector pose at the same time, the minimum of both fractions c_T and c_R has to be applied (which eventually leads either to lower linear or angular velocity than desired - at the benefit of a smoother motion). So finally ΔT can be given in the decoupled representation with $c = \min[c_R, c_T]$

$$\Delta T = \delta T = \langle \boldsymbol{q}_\delta, \boldsymbol{t}_\delta \rangle = \langle \boldsymbol{q}_\Delta^c, c\,\boldsymbol{t}_\Delta \rangle. \tag{4.21}$$

Up to this point an approach for generating motion yielding constant end-effector velocity in operational space, i.e., the discrete equidistant interpolation in $SE(3)$, has been presented. In the following sections, an extension to this approach is derived. This is necessary, because out of a standby configuration with zero speed, physical robots in general do not allow for application of a constant speed towards arbitrary directions, as this would require infinite acceleration and most likely damage the hardware.

Typical approaches therefore apply so-called advanced *velocity profiles* for generating a path. The velocity profile for the approach described in this section is called a *constant velocity* profile. It can be seen in the uppermost diagram of Figure 4.2. The diagrams below show advanced approaches: in the center a *trapezoidal* profile is depicted and a S-ramped profile as an approximation of a *minimum-jerk* profile can be found in the bottom. These profiles result in significantly smoother trajectories which typically decrease the energy consumption and increase the life-time of a robot [de Michieli et al. 2008, Huber et al. 2009] – and make the motion look more natural [Flash and Hogan 1985].

Trapezoidal Velocity

Trapezoidal velocity profiles are named that way, because if one looks at the velocity value on a time chart, the resulting curve looks like a trapezoid (see Figure 4.2). Considering the generation of a trapezoidal velocity profile on a motion path, the

Summary 4.1 – **Compositional Constant Velocity Profiles**

- Considers the current position and angle configuration, but no history.
- Moves the end-effector towards a target at a constant velocity.
- Does not take care of hardware limits in configuration space.

1. Compute the interpolating transformation ΔT between the current pose and the target pose.
2. Compute the "scaling" factors c_T and c_R according to the desired angular and linear velocities and subject to the hardware update rate.
3. For smooth motions (orientation and translation are completed at the same time) compute the intermediate pose δT with the minimum of both scaling factors.
4. Apply the next angle configuration from the pose using inverse kinematics formula $\boldsymbol{\Delta\theta} = J^{-1}\boldsymbol{\Delta p}$.

linear relation, i.e., direct proportionality, of spatial displacement and time can be exploited. Characteristically, the acceleration for a trapezoidal velocity profile is constant (at least within each phase of the motion). This means we can distinguish three periods for moving along a path.

1. A constant acceleration phase with increasing speed and thus with increasing absolute pose increments
$$|\boldsymbol{r}_\Delta^{t+1}| > |\boldsymbol{r}_\Delta^{t}| \quad \wedge \quad |\boldsymbol{t}_\Delta^{t+1}| > |\boldsymbol{t}_\Delta^{t}|.$$
2. The zero acceleration phase (constant speed) with equal absolute pose increments
$$|\boldsymbol{r}_\Delta^{t+1}| = |\boldsymbol{r}_\Delta^{t}| \quad \wedge \quad |\boldsymbol{t}_\Delta^{t+1}| = |\boldsymbol{t}_\Delta^{t}|.$$
3. A constant deceleration phase with decreasing speed and decreasing absolute pose increments
$$|\boldsymbol{r}_\Delta^{t+1}| < |\boldsymbol{r}_\Delta^{t}| \quad \wedge \quad |\boldsymbol{t}_\Delta^{t+1}| < |\boldsymbol{t}_\Delta^{t}|.$$

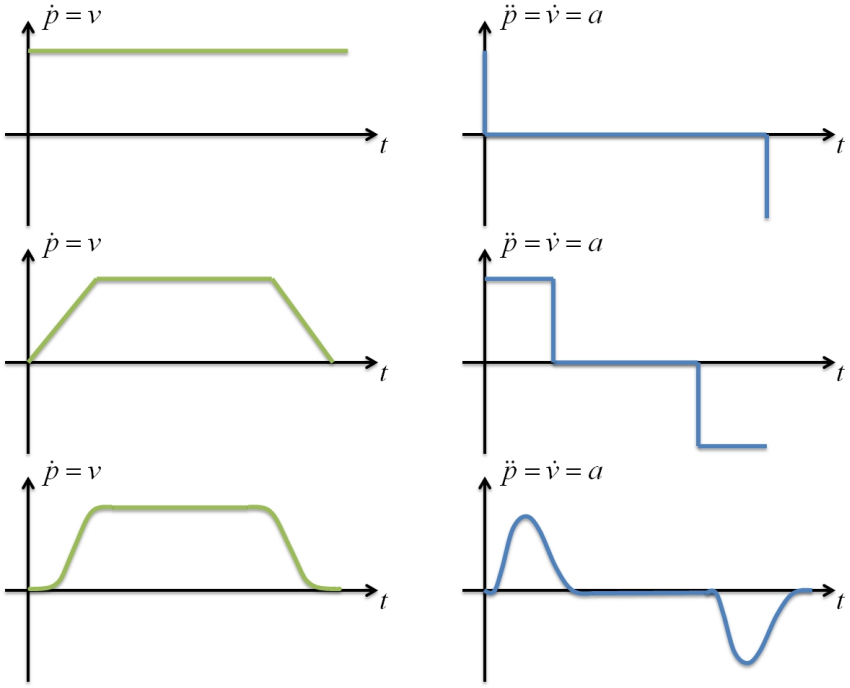

Figure 4.2 *Common targeted velocity profiles for path generators. Upmost: constant velocity profile; center: trapezoidal profile; bottom: S-ramped profile.*

A constant acceleration parameter (velocity gradient) is usually chosen in advance, it will be called a in this section. Due to the simple relation of speed v, time t, and distance s, i.e., $v = \frac{s}{t}$, a whole set of formulas for application of trapezoidal velocity profiles in compositional systems can be derive relatively straightforward.

First, we state that it is possible to determine a distance in terms of translation and rotation from the current pose to the target and using the linear relation between time and spatial distance, a difference in terms of time from the current point to the end of the motion. We need to know this, in order to compute the point when to stop accelerating on the trajectory, as we have reached the desired speed. Furthermore, we can use this relation to compute precisely when the deceleration phase has to be started in order to smoothly reach the target at zero speed, which is a differential equation in $SE(3)$

$$f(\boldsymbol{p}) = \boldsymbol{p}^t + \frac{d}{dt}\boldsymbol{p}^t \delta t, \tag{4.22}$$

but a linear equation in the differential domain $se(3)$. In the differential domain, regarding discretized time steps $t \mapsto t + 1$, we give the equation

$$|m^t(\boldsymbol{\delta}_T, \boldsymbol{\delta}_R)| = |m^{t-1}(\boldsymbol{\delta}_T, \boldsymbol{\delta}_R)| + \alpha^t a \tag{4.23}$$

with a *constant* increment a on the pose increment (the acceleration parameter) and a signum factor α. The factor α is used to distinguish between the three phases and given by

$$\alpha = \begin{cases} +1 & \text{for acceleration,} \\ 0 & \text{for constant velocity,} \\ -1 & \text{for deceleration.} \end{cases} \tag{4.24}$$

Within the acceleration period, we can thus write

$$|m^t(\boldsymbol{\delta}_T, \boldsymbol{\delta}_R)| = |m^{t-1}(\boldsymbol{\delta}_T, \boldsymbol{\delta}_R)| + a. \tag{4.25}$$

Within the deceleration period on the other hand, the constant a only flips the sign

$$|m^t(\boldsymbol{\delta}_T, \boldsymbol{\delta}_R)| = |m^{t-1}(\boldsymbol{\delta}_T, \boldsymbol{\delta}_R)| - a, \tag{4.26}$$

while $a = 0$ within the constant velocity period. Also this means after the desired velocity v is reached, the increment's absolute value $|m^t(\boldsymbol{\delta}_T, \boldsymbol{\delta}_R)|$ remains constant.

The difficult part is to determine the exact point when to stop acceleration and maybe even more important, when to start deceleration. Based on Equation (4.8), i.e., $\boldsymbol{p}^t = \sum^t \boldsymbol{\delta p}^i + \boldsymbol{p}^{home}$, utilizing induction yields the equation for the distance s^t (in terms of Euclidian distance or absolute angle) traveled by the end-effector after some discrete time $t = nu$ given in terms of the number of update steps n at an update rate u

$$\begin{aligned} s^t &= s^0 + s^1 + s^2 + \ldots + s^{n-1} + s^n \\ &= v^0 u + (v^0 + \alpha^0 a)u + (v^1 + \alpha^1 a)u + \ldots + (v^{n-1} + \alpha^n a)u. \end{aligned} \tag{4.27}$$

This is still a general form. Yet, considering being within one of the periods of acceleration, deceleration, or constant velocity, the equation can be simplified to[6]

$$\begin{aligned} s^t &= v^0 u + (v^0 + 1a)u + (v^0 + 2a)u + \ldots + (v^0 + na)u \\ &= v^0 u(n+1) + au \sum_{i=1}^{n} i \\ &= v^0 u(n+1) + au \frac{n(n+1)}{2} \\ &= (v^0 + \frac{n}{2}a) \cdot u(n+1). \end{aligned} \tag{4.28}$$

[6] Equation (4.28) only shows the derivation for $\alpha = +1$, i.e., the acceleration period. Deceleration and constant velocity work analogously, and evaluate to $s^t = (v^0 - \frac{n}{2}a) \cdot u(n+1)$ for deceleration and $s^t = v^0 nu$ for constant velocity.

The above equation holds for both translations (setting $v = v_T$) and rotations (setting $v = v_R$). Thus, for subsequent equations the notation s_T^t is used to concretize for translations and s_R^t for rotations respectively.

If at any point while the end-effector is moving on a trajectory the distance measure s^t with $n = \frac{t}{u}$ is evaluated, the signum constraint for distinguishing phases, i.e., for acceleration, deceleration, or constant velocity, is given for that point as

$$\alpha^t = \begin{cases} +1 & \text{for} \quad s_T^n u^{-1} < v_T \quad \wedge \quad s_R^n u^{-1} < v_R, \\ -1 & \text{for} \quad |s_T^t| \leq |\boldsymbol{\Delta}_T| \quad \vee \quad |s_R^t| \leq |\theta_{\boldsymbol{\Delta}_R}|, \\ 0 & \text{else,} \end{cases} \tag{4.29}$$

where in any case $v_T^0 = v_R^0 = 0$ is set for evaluations of s^t, because the desired speed at the end (or beginning) of the movement is zero. This simplifies Equation (4.27) to

$$\begin{aligned} s_T^t &= a_T \cdot u \frac{n(n+1)}{2} \\ s_R^t &= a_R \cdot u \frac{n(n+1)}{2}. \end{aligned} \tag{4.30}$$

For infinitesimally small $u \to 0$, i.e., the time derivative, this yields an exact solution. On the other hand, whenever $u \gg 0$, the actual movement of the end-effector might suffer from exaggerated velocities, overshooting, and / or jerking around the target pose. To avoid this, α can be made a real $\in [-1, 1]$ in order to scale the last a before stopping acceleration or deceleration to yield a smaller increment than evaluated by the constraint from above.

To apply a trapezoidal velocity profile for path generation, the approach described in Summary 4.2 is recommended.

Minimum Jerk and *S*-ramped Profiles

Within the last section, trapezoidal velocity profiles have been shown to work in compositional systems, as the acceleration and deceleration parameter is a constant within the velocity domain. In this section, this constant a is made a variable leading to an even smoother path. The path generator is hence required to produce pose increments for a path that is differentiable considering second and third order time derivatives of location, in addition to minimizing the variations of the third order term.

Neville Hogan showed in 1984, that these trajectories in fact comply with the ones biological systems generate when they reach for a target. He found evidence for that

Summary 4.2 – **Point-to-Point Motion with Trapzeoidal Velocity Profiles**

A step-by-step quick reference to application of the trapezoidal velocity profile is:

❶ Choose a desired maximum speed v_{max}, which will be the absolute speed of the end-effector within the constant velocity phase.

❷ Choose an acceleration parameter a according to the update rate of the robot. Example: if the update rate is $u \approx 0.007\,s$, as for the Mitsubishi RV6 robots, and v_{max} shall be reached after $0.7\,s$, the acceleration parameter a can be derived from $n = tu^{-1} = 100$ and $v_{max} = v_0 + na$ where $v_0 = 0$, thus

$$a = \frac{v_{max}}{n} = 0.01\,v_{max}.$$

❸ Within every cycle during the movement, evaluate the constraints for α^t with the distance measures s_T^t and s_R^t. If one of the constraints changes, a phase transition occurs.

❹ Compute the increment according to $|m^t(\boldsymbol{\delta}_T, \boldsymbol{\delta}_R)| = |m^{t-1}(\boldsymbol{\delta}_T, \boldsymbol{\delta}_R)| + \alpha^t a$, which in particular requires rescaling m^{t-1} within the acceleration and deceleration phases.

when observing metabolic energy inputs to antagonistic muscles. From the observations he concluded that minimizing the third order time derivatives of location (also called *jerk*) efficiently minimizes the energy needed to move along a trajectory [Hogan 1984]. To express the cost C, the formula for the magnitude of third order variations during a movement with final position at time t_f is given by[7]

$$C = \frac{1}{2}\int_0^{t_f} \left(\frac{d^3x}{dt^3}\right)^2 dt \tag{4.31}$$

This is the sum of squared jerks along the trajectory $x_0 \to x_f$. [Flash and Hogan 1985] then showed how to model the trajectory generator of the central nervous system mathematically, i.e., stated that the system has to minimize Equation (4.31) for a given target position. Flash and Hogan showed, that after minimization the

[7] The equations in this section adopt the notation of Hogan and Flash. Only at the end of the section the notation is adjusted in order to transfer the results.

optimal trajectory, for example for a hand moving towards the final location x_f with zero initial and target velocities and with respect to time, is given by

$$x(t) = x_0 + (x_0 - x_f)\left(15\frac{t^4}{d^4} - 6\frac{t^5}{d^5} - 10\frac{t^3}{d^3}\right), \tag{4.32}$$

where d is the desired duration of the movement. After some rearrangement, the time derivatives of first, second and third order can be written as

$$\begin{aligned}
\dot{x}(t) &= 30(x_f - x_0)\left(\frac{t^2}{d^3} - 2\frac{t^3}{d^4} + \frac{t^4}{d^5}\right) \\
\ddot{x}(t) &= 60(x_f - x_0)\left(\frac{t}{d^3} - 3\frac{t^2}{d^4} + 2\frac{t^3}{d^5}\right) \\
\dddot{x}(t) &= 60(x_f - x_0)\left(\frac{1}{d^3} - 6\frac{t}{d^4} + 6\frac{t^2}{d^5}\right).
\end{aligned} \tag{4.33}$$

In FIGURE 4.3 one can see that the velocity has its maximum at $t = \frac{1}{2}d$ and becomes zero at $t = 0$ and $t = d$. This can be shown using the second and third order time derivatives. In particular, we find that $\ddot{x}(t)$ has extrema at these points in $t \in [0, d]$, but only $\dddot{x}(\frac{1}{2}d) < 0$. So the point of maximum velocity is at $t = \frac{1}{2}d$.

The velocity at some point $\dot{x}(t)$ can be simplified using the average velocity $v_{av} = (x_f - x_0)d^{-1}$,

$$\begin{aligned}
v(t) &= \dot{x}(t) = \frac{d}{dt}x(t) \\
&= 30(x_f - x_0)\left(\frac{t^2}{d^3} - 2\frac{t^3}{d^4} + \frac{t^4}{d^5}\right) \\
&= (x_f - x_0) \cdot 30\frac{t^2}{d^5}(d^2 - 2dt + t^2) \\
&= \frac{30(d-t)^2t^2}{d^4}v_{av},
\end{aligned} \tag{4.34}$$

and a relation between the maximum velocity and the average velocity can be derived[8] by setting $t = \frac{1}{2}d$,

$$v_{max} = v(\frac{1}{2}d) = \frac{30 \cdot 0.0625\, d^4}{d^4}v_{av} = 1.875\, v_{av}. \tag{4.35}$$

Since an expression for the absolute increment on the trajectory for a given point in time t regarding an update rate u can be given as

$$v(t + u) = \frac{\delta_u x(t)}{u}, \tag{4.36}$$

[8] Psychophysical experiments, e.g., [POZZO ET AL. 1998], reveal that human reaching movements typically have a ratio of about 1.75 (apart from whole-body balancing effects), which is the reason why minimum jerk profiles are believed to best resemble human motions.

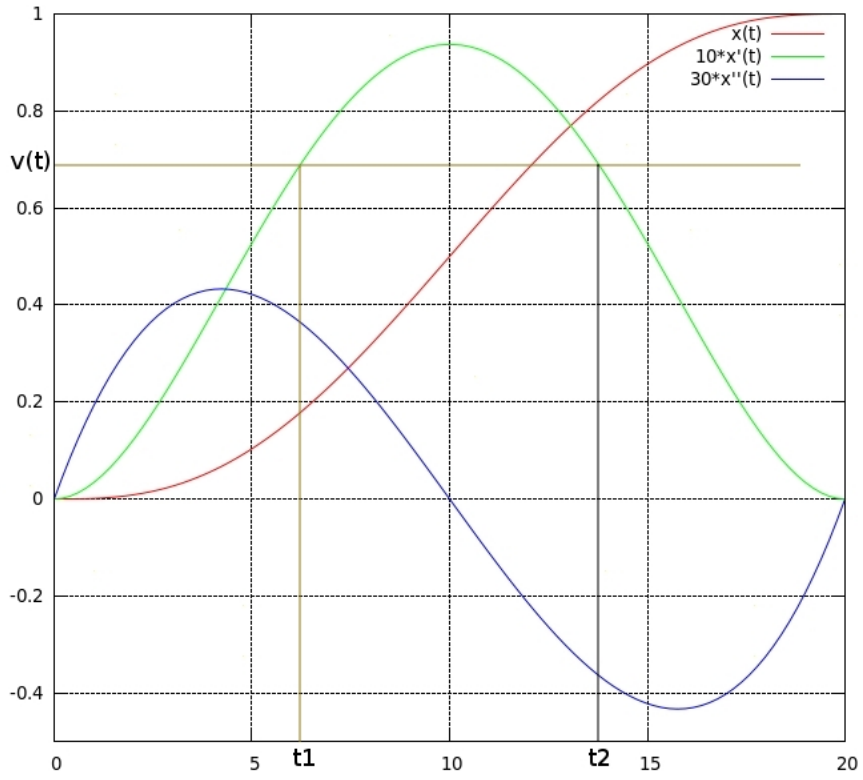

Figure 4.3 *Red sigmoid: minimum jerk trajectory for moving an end-effector from a position 0 to 1 in 20 time units; green curve: velocity profile; blue curve: acceleration profile. If only the velocity is given, the end-effector position is ambiguous, as it may be in the acceleration phase t_1 or deceleration phase t_2 of the movement.*

the compositional form of the trajectory can be written as

$$x(t+u) = x(t) + \delta_u x(t) = x(t) + v(t+u)u. \tag{4.37}$$

The problem is, that using the compositional form, usually the actual value of t is unknown. Yet, referring to FIGURE 4.3, there are only two solutions for t if only the last increment $\delta_u x(t-u)$ and thus $v(t)$ are known. This means, for any given $v(t)$ under the assumption of a fixed duration for the movement d, EQUATION (4.34) can be solved

$$\begin{aligned} v(t) &= \frac{30(d-t)^2 t^2}{d^4} v_{av} \\ \frac{v(t)d^4}{30\, v_{av}} &= (d-t)^2 t^2 \end{aligned}$$

$$\begin{aligned} \pm d^2\sqrt{\frac{v(t)}{30\,v_{av}}} &= (d-t)t \\ 0 &= t^2 - d\,t \pm \sqrt{\frac{v(t)}{30\,v_{av}}}d^2 \end{aligned} \tag{4.38}$$

yielding four solutions

$$t_{1,2} = \frac{d}{2} \pm \frac{d}{2}\sqrt{1-4\sqrt{\frac{v(t)}{30\,v_{av}}}} \quad \text{and} \quad t_{3,4} = \frac{d}{2} \pm \frac{d}{2}\sqrt{1+4\sqrt{\frac{v(t)}{30\,v_{av}}}}. \tag{4.39}$$

Of these four solutions only t_1 and t_2 evaluate to numbers within the interval $[0, d]$ of interest, because

$$\max_{v(t)}\left(4\sqrt{\frac{v_{max}}{30\,v_{av}}}\right) = 4 \cdot 0.25 = 1 \quad \text{and} \quad \min_{v(t)}\left(4\sqrt{\frac{0}{30\,v_{av}}}\right) = 0. \tag{4.40}$$

One of the solutions lies within the acceleration and one within the deceleration period. Hence, it has to be decided in which phase of the movement the end-effector currently is. The constraints to be evaluated with this respect are

$$t = \begin{cases} t_1 = \frac{d}{2} - \frac{d}{2}\sqrt{1-4\sqrt{\frac{v(t)}{30\,v_{av}}}} & \text{if} \quad |x_f - x(t)| > \frac{1}{2}|x_f - x_0|, \\ t_2 = \frac{d}{2} + \frac{d}{2}\sqrt{1-4\sqrt{\frac{v(t)}{30\,v_{av}}}} & \text{else,} \end{cases} \tag{4.41}$$

where t_1 to the acceleration period period and t_2 corresponds to the deceleration respectively.

In simple words this means, wherever on the minimum jerk path the end-effector is at the moment, if the current velocity (from the last increment) and the distance to the target are known, the next increment can be computed analytically. This works under the assumption that a single motion always takes the same time d, which may not be the case when target poses are relocated during the movement (see Section 4.1.2).

In Hogan's original Equation (4.32), x_0 and x_f are scalar values, so $x(t)$ is effectively a function of distance with respect to time. For trajectory generation in $SE(3)$, i.e., 3D space with six degrees of freedom, this has to be extended. Luckily, this can be easily done using the same methodology as in the previous section. Only the absolute value of the increments have to be set according to the formalism derived in this section

$$|m^t(\boldsymbol{\delta}_T, \boldsymbol{\delta}_R)| = \delta_u x(t) = v(t+u)u. \tag{4.42}$$

Again, $v(t)$ refers to a composition of the linear Cartesian velocity $v_T(t)$ and the angular velocity $v_R(t)$. The above thus has to be evaluated twice, once for the linear displacement $\boldsymbol{\delta}_T$ and once for the angular displacement $\boldsymbol{\delta}_R$, each of which again is a vector with three components.

Summary 4.3 – **Minimum Jerk Velocity Profiles**

❶ Choose desired duration d for the motion.

❷ Choose the maximum for linear and angular velocities. The average velocities are then given by rearranging EQUATION (4.35) to

$$v_{av} = \frac{1}{1.875}\, v_{max}.$$

❸ Beginning at $v(t+u) = v(0+u)$, calculate the $\delta_u x(t)$ for linear and angular displacements according to the minimum jerk velocity profile.

❹ If t is unknown and thus it is not clear whether the system is in the acceleration or deceleration phase, but only the current velocity is known, the actual corresponding $t = t_1 \vee t_2$ can be computed using EQUATION (4.41) and the absolute of the next increment can be written as

$$|m^t(\boldsymbol{\delta}_T, \boldsymbol{\delta}_R)| = v(t+u)u.$$

❺ At each point map the absolute value of the displacements to actual increments in $SE(3)$ to be applied.

4.1.2 Operational Space Freely Movable Targets

The results from above, i.e., how to compute a straight path when moving towards a stationary target in $SE(3)$, is of great use in classical robotic environments. However, complex dynamic environment scenarios require a robotic system to be capable of recomputing a path at any time while preserving sufficient smoothness during a motion. Obviously applications for such a realtime path generator capable of changing the heading direction of the end-effector at any instant of time are manifold. Sensor-based servoing and active perception (moving towards a tar-

get computed from sensory input data), realtime obstacle avoidance (for mobile or dynamic environments), or force-torque control are some popular examples.

As mentioned before, realtime modifications on a path have to obey certain rules, i.e., the resulting curve must satisfy smoothness constraints with respect to pose, velocity, acceleration, or even higher order terms. Mathematically, this means the path-curve must be differentiable at least n times, if $(n-1)$-th order derivative is to be continuous. Practically speaking, if a path shall be smooth (where path refers to the curve and orientation variation the end-effector describes in operational space), its generating function $f(t) = \boldsymbol{p}^t$ has to be differentiable at any time. If then also the velocity profile should be smooth, also $\frac{d}{dt}f(t)$ must be differentiable, and so on for acceleration and higher order terms.

For generating a n-th order differentiable path, the system needs to remember n past poses (a motion history) or the n-th order derivative values at the current pose. For example revealing a compositional trapezoidal velocity profile is possible, if the system only stores the last velocity values (first order terms). For a continuous acceleration profile up to second order terms have to be available, and for differentiable acceleration profiles third or higher order terms need to be present.

Depending on n, the number of higher order terms considered, the presented approach ensures that the constant $(n+1)$-th order term in worst case flips the sign at one single update cycle during the movement, while the absolute value of the increment is preserved. This makes the algorithm applicable to a practical robot motion generator, as implemented as part of this work.

For simplicity, the realtime path generation system shown in the next paragraphs considers only up to third order terms. The generated path thus always has a differential acceleration and velocity profile and yields a so-called S-ramped profile. This serves as a practical approximation to the minimum jerk profiles for stationary targets described in the last section.

The general idea of the algorithm comprises a two step update at each robot update cycle (see FIGURE 4.4).

1. This first update recomputes the current intermediate target according to the desired global target. The global target can be modified at any time, by a higher level control unit (e.g. an obstacle avoidance, or visual servoing system).

2. The second update recomputes the next pose increment according to the current and desired n-th order term, which is computed according to the desired velocity profile.

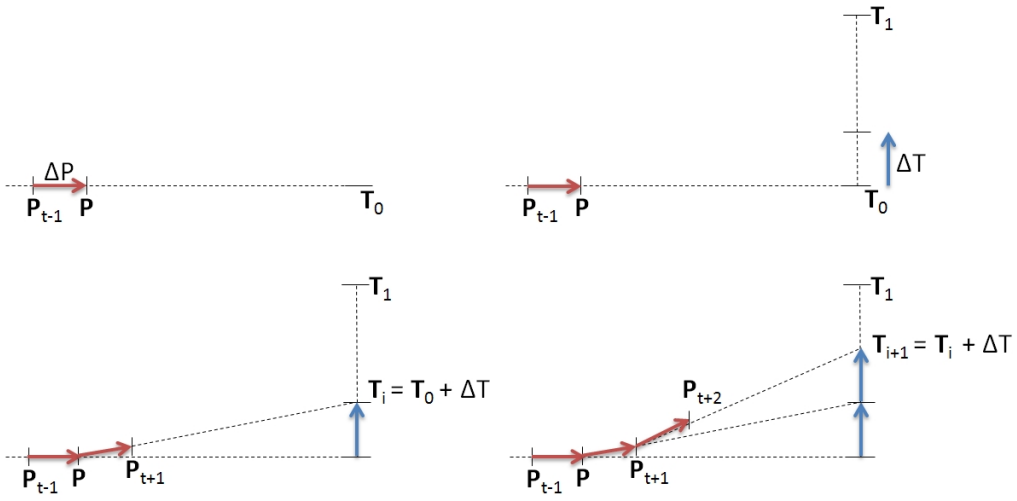

Figure 4.4 *Realtime path generation algorithm details for a linear motion in a 2D plane for first order terms only (yielding a constant velocity profile).*

In other words, whenever the desired global target changes, an intermediate target travels from the current global target towards the new one. The pose increment, i.e., the next desired end-effector pose is then computed towards this intermediate instead of to the new global target. The target transition (blue) also uses the same (n)-th order velocity profile as the actual pose transition (red).

The next sections briefly introduce the approach to incrementally complex scenarios, i.e., constant, and higher-order continuous and differential velocity profiles.

Constant Velocity

Realtime constant velocity path generation refers to the problem of computing an increment on the current end-effector pose such that the absolute value of the linear and angular deltas are constant whenever the robot moves

$$\forall i, j: \quad |\boldsymbol{v}(t_i)| = |\boldsymbol{v}(t_j)| \quad \wedge \quad |\boldsymbol{\omega}(t_i)| = |\boldsymbol{\omega}(t_j)|. \tag{4.43}$$

As mentioned above, the algorithm ensures this for any velocity increment (constant velocity refers to $n = 1$, so first order terms are considered). For a single increment, at each change of a target the worst case ($\epsilon \to 0$) is that

$$\boldsymbol{v}(t_i) = \lim_{\epsilon \to 0} [-(1-\epsilon)\boldsymbol{v}(t_{i-1})] = -\boldsymbol{v}(t_{i-1}), \tag{4.44}$$

i.e., the components of the velocity vector flip their sign. For clarification, the constant velocity worst case is detailed in FIGURE 4.5. The worst case occurs,

whenever the new (intermediate) target requires a immediate backward turn with respect to heading direction, or a sudden reflection of orientation (or both). The figure shows the situation for a 2D path (single dimension) only. However, the worst case of course generalizes to any pair of 3D displacement or 3D rotation dimensions in $SE(3)$.

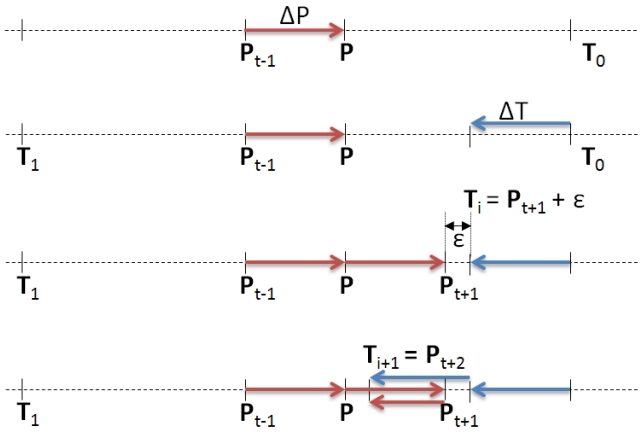

Figure 4.5 *Worst case scenario for the realtime path generation algorithm: reflection of heading direction and / or desired orientation.*

Higher Order Profiles

One can easily infer from the constant velocity case described above, that the algorithm generalizes for higher order velocity profiles, i.e., where continuity or differentiability on higher order terms (acceleration, jerk) are required. Such profiles are depicted at the bottom of FIGURE 4.2 and in FIGURE 4.3. Here, not the velocity may flip the sign on a reflection, but the n-th order term (acceleration, jerk, etc.) does instead. For example, in the case of a trapezoidal profile, the trapezoid of the velocity profile in the worst case degenerates to a wedge-like profile, i.e., the constant velocity period vanishes, as the acceleration flips the sign. In even higher order profiles, this flipping propagates to higher order derivatives analogously. Thus, the continuity required from such velocity profiles can still be maintained. The constraint that has to be satisfied is, that the first update step, i.e., the target transition must obey the same velocity profile than the desired motion trajectory.

Only the path generation system violates the smoothness constraint, if the desired velocity profile is $n \to \infty$ times differentiable. In practice though, such profiles are not applied normally, as they produce high computational costs. As in a realtime target motion scenario the final motion duration is not known, because the target

is allowed to move freely and constantly, "smoothed" trapezoidal profiles normally serve as an approximation to exact high order or minimum jerk profiles. Such profiles (shown e.g. in FIGURE 4.2) are sometimes called S-ramped or S-curved profiles [NGUYEN ET AL. 2008, THOMAS 2003].

Here the path generation system, instead of computing an exact period of acceleration and deceleration, where v_{max} is only reached at a single cycle, introduces a period of constant v_{max}, whenever the desired maximum speed is reached, while the acceleration and deceleration periods are smoothed (S-ramped) according to some predefined default motion duration (e.g., one second for the system implemented in this work).

4.1.3 Application: a Low-Level Robot Controller

The low-level control and data acquisition units of the scenario in SECTION 5.1.1 directly connect to hardware interfaces. In this scenario these hardware interface comprise the robot's realtime interface (i.e., the robot controller), the servo gripper controller board and the camera devices.

The robot control unit receives activation features and interprets them as Cartesian targets (for debugging purposes also direct joint angle updates are possible) from the activation layer units and computes a possible path in realtime resulting in a sequence of angle increments on the current configuration (compositional update) to reach the target as described in above. Complying with realtime constraints, the unit is timed (see in particular SECTION B.1.4) and apart from timing errors, it emits an error event, as soon as a singular configuration would be reached and triggers a recovery strategy within the encapsulated path generation unit or stops the robot in the worst case. Higher-level control units then have to modify the joint angles in order to recover from the error state. Also the units allows for sharing the current angle configuration on a data channel for example for visualization of the robot motion in a simulation environment (see FIGURE 4.6).

4.2 Perception and Attention

Activation in the perception domain has not yet been covered with respect to continuous transformation groups. First, SECTION 4.2.1 a relatively simple perception system is described that uses bottom-up stimulus and top-down feedback for generation of activation corresponding to regions of interest in the perceptive field. SECTION 4.2.2 shows, how the bottom-up approach can be extended to produce ac-

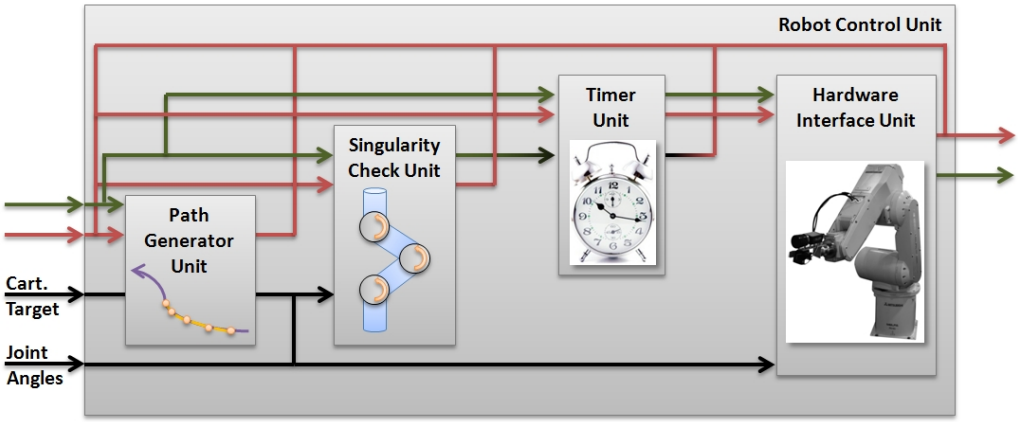

Figure 4.6 *Schematic overview of the time-constrained robot control unit.*

tivation features that can be pushed into the association layer. Then Section 4.2.3 combines preliminary work with the bottom-up approach in order to integrate incoming activation feedback into a selective attention mechanism. The remaining sections discuss applications in two scenarios: recognition and tracking of interaction partners through visual observation and workspace surveillance in an assembly system.

4.2.1 Preliminary Work

In the case of visual processing and a calibrated setup previous work (e.g., [Müller and Knoll 2009a]) utilizes a 2D intensity image as a space for representing activation directly. For the purpose of attention evaluation higher-level environment knowledge, such as robot end-effector or limp positions, tracked object poses and others, is projected into the visual field of view. An attention or saliency map is then computed as a gray-scale activation map in the first place. Second, by addition, this activation map is merged with the bottom-up regions of attraction, also computed as a gray-scale activation map. After applying a suitable threshold as a final step the result is a binary image as shown in Figure 4.7.

This is totally sufficient for attention applications based on calibrated mono camera vision, but it may not generalize for other, more complex applications based on activation. Furthermore, in particular where multiple types of sensors have to be used concurrently with complex actuators, it is certainly not possible to represent perception and action effects only in the perceptive space. Thus, a more general representation is needed.

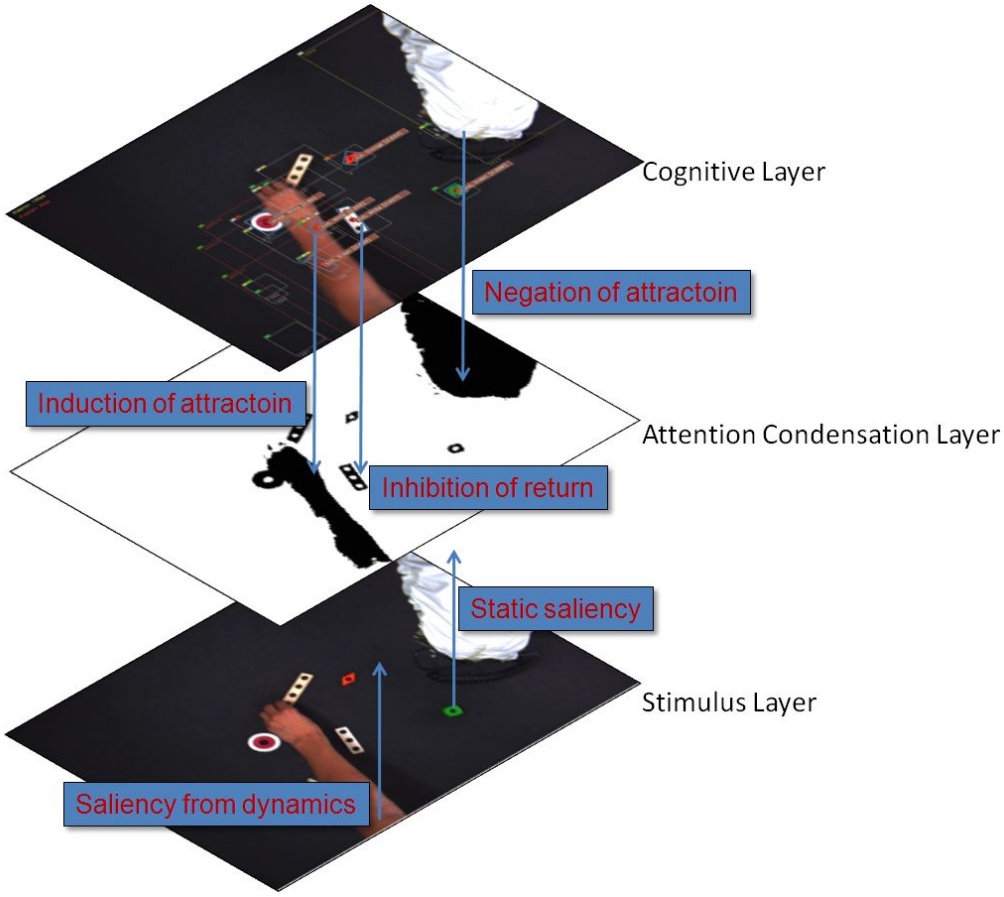

Figure 4.7 *An attention condensation layer represented in the visual field – a 2D transformation space projection.*

4.2.2 Activation from Bottom-Up Stimulus

Traditional computer vision systems, either for surveillance tasks with fixed cameras, robot vision on mobile platforms or any other kind of visual processing suffer from an enormous computational complexity. The main reason for this is, that these traditional systems utilize complete analysis approaches: after acquiring input data from its camera(s), the system has to process the huge amount of data and analyze all of it regardless of the significance to the current task or environment. Now, the basic idea is to apply an attention-based information filter in order to reduce the amount of input data and only perform further analysis (for example the object detection algorithm from Section 5.4.1) on the rest. This residual is what we call the regions of interest (ROIs).

In particular one aspect of an attention-based visual information filter is ideally suited for use with continuous transformation groups: the concept of dynamic

bottom-up attraction introduced in SECTION 2.1.3. Consider for example an object moving in the visual field of view. According to neuroscientific findings especially such moving objects (or uniformly moving clusters) attract attention to subjects.

The idea here is to extract moving features from pairwise consecutive images in a sequence. For example the *Lukas-Kanade* feature based optical flow algorithm [LUCAS AND KANADE 1981, LUCAS 1984, BOUGUET 2000] would be suitable for this task (see FIGURE 4.8). A such feature is a quadruple

Figure 4.8 *Feature based optical flow result: motion field for a moving object and a mean transformation generator projected back into the image.*

$$F = \{x, y, dx, dy\}, \tag{4.45}$$

where x, y are the coordinates in the image and dx, dy are pixel displacements respectively.

Considering the image plane as the transformation space of reference the features detected from the algorithm represent a 2D projection of the actual 3D transformations that occur on voxels in the real world. In this case the transfer to group theory's notation is simple, as each feature directly contains the scale values $s_x = dx$ and $s_y = dy$ for the generators according to EQUATION (2.21)

$$dx = \exp(s_x \cdot G_x) = \exp(s_x \cdot \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}), \tag{4.46}$$

and for dy analogously, in the Lie group for 2D translations represented as $\mathbb{R}^2$. The algebra elements then are

$$\boldsymbol{s} = (s_x, s_y)^T. \tag{4.47}$$

So regions of uniform motion (regions of interest) can be found simply by evaluating

$$\forall i, j \in n: \quad w_\times(\boldsymbol{s}_i \times \boldsymbol{s}_j) + w_-|\boldsymbol{s}_i - \boldsymbol{s}_j| \approx 0, \tag{4.48}$$

on any n features in a local neighborhood within the input image. This weighted sum condition works, because the cross-product $\boldsymbol{s}_i \times \boldsymbol{s}_j$ evaluates to zero in the "best" case of optical flow features pointing to the same direction and respectively also the absolute value of the difference vector of each two features becomes zero if they have the same length. In practice, because of noise the direction weight $w_\times$ would typically be greater than the displacement weight, while $w_\times + w_- = 1$ always has to be satisfied.

In FIGURE 4.9 two more examples, this time on deformable linear structures, are shown. The region of interest extracted with the algorithm is indicated with a red outline. Furthermore, in the figure arrows of the same color refer to activation features that correspond significantly (by activation of Lie algebra elements and spatio-temporal context). Note that the region building process also takes the position information into account. It also finds a common region of interest by merging, where spatially overlapping intermediate regions occur. Intermediate regions are not explicitly shown in the figure, but one can easily infer their outline, as they would be regions only comprising features of the same color.

According to [DRUMMOND AND CIPOLLA 2000], motion of a 2D planar region can be correlated to activation, i.e. a Lie algebra element of $se(3)$ by projection from the general affine group of transformations of two dimensions $GA(2)$ subject to the image plane under the assumption of weak perspective transformation (otherwise $P(2)$ needed to be used to model strong perspective distortions). However, as the regions have a non-rigid area boundary, in the current system a calibrated setup is needed for creating activation features for the association layer. Furthermore, on a mono-camera setup the feature lacks at least one dimension, as the observable activation is at most a projection of a transformation in $SE(3)$. Nevertheless, this is not in general a constraint, as features originating from different sensors can be combined in the association layer.

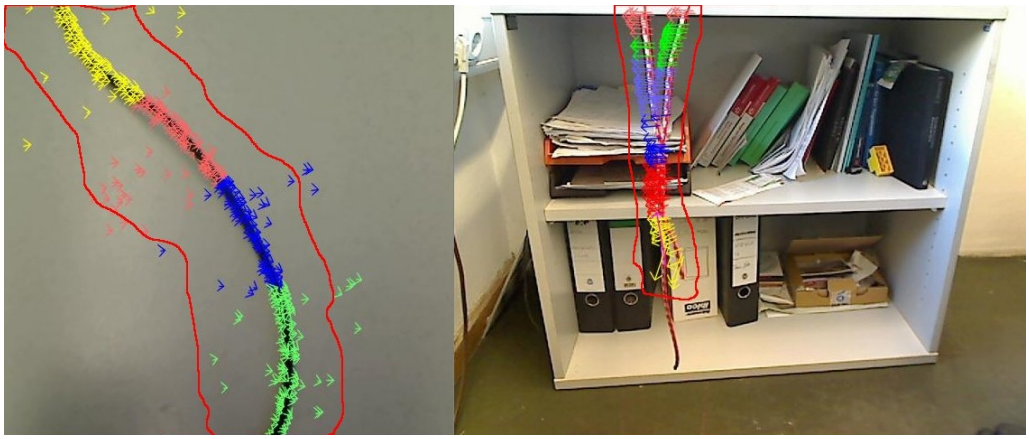

Figure 4.9 *Linear deformable structures moving within the visual field. The red outline indicates the ROI. The algorithm clusters overlapping intermediate regions (which would consist only of features with the same color).*

4.2.3 Selective Activity With Top-Down Feedback

In biological attention systems, according to recent neuroscientific findings (as introduced in Section 2.1.3), two main mechanisms accomplish a low-level information filter: *bottom-up* attraction originates directly from sensory devices and affects the attentive activation within certain regions of the perceptive field (see above); and *top-down* or feedback mechanisms inhibit or enforce activation based on higher-level processes and context knowledge. Only regions with significant activation are being payed attention to, i.e. are in a computational sense worth further processing. In this section the integration of the second aspect into the above approach is elaborated on in greater detail.

In order to integrate bottom-up and feedback effects into an attention condensation mechanism, the attention effects have to be projected into a common transformation representation. In addition to the association layer, where activation features of $se(3)$ are combined and propagated to higher-level and actuator modules, in the perception system an association of activation projected into a subspace corresponding to the image plane takes place.

Utilizing the Lie algebra vector space of this perceptive tranformation space, attention, i.e., regions of high activation in the perception domain can be defined easily and then be pushed to the association layer for further processing of sensory input data. As a result, the sensorimotor robotic system essentially consists of two association subsystems. One for sensorimotor association (see Section 4.3), i.e. the

aforementioned activation association layer comprising features of $se(3)$, and an additional association layer for visual attention (which works on projected features of $se(3)$ and bottom-up activation).

As regions of attention only make sense within a spatio-temporal context, from the set of activation features only the ones referring to a relevant context are selected for determining the level of attention in a region. The selected features may have positive (excitatory) or negative (inhibitory) weight, depending on the type of influence they represent. Consider for example a mono-camera setup as shown in FIGURE 4.10. At some point in time the level of attention regarding the complete perceptive field is simply the accumulated activation, i.e., the sum of all feature descriptors,

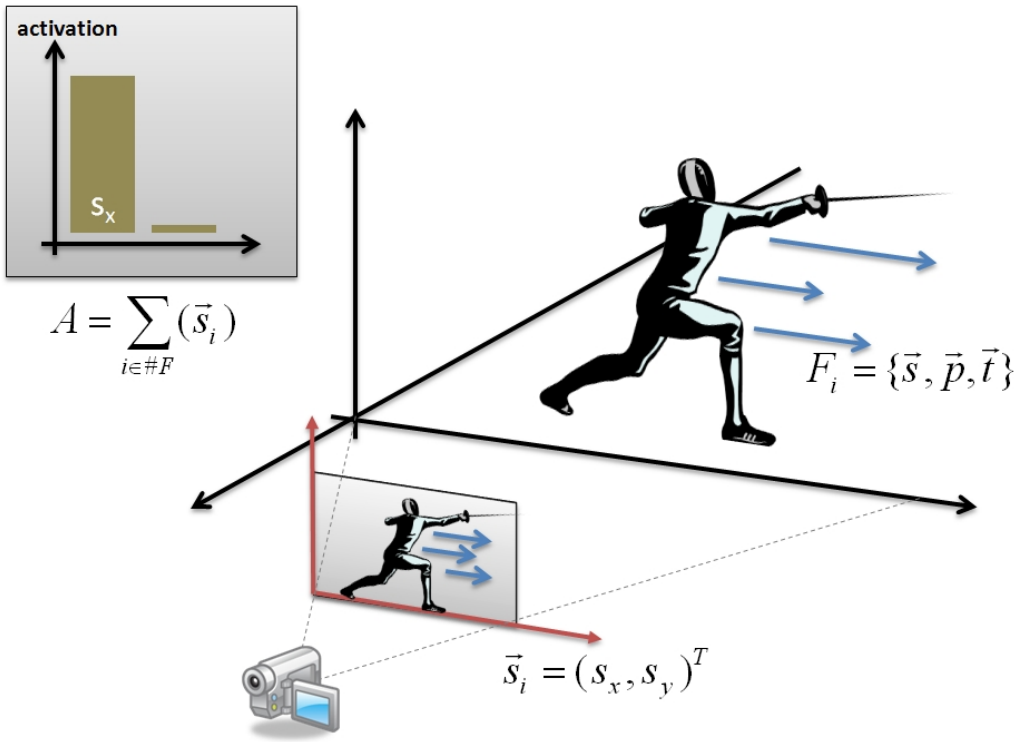

Figure 4.10 *2D mono-camera setup: features in sensor space projection and accumulated activation in the perceptive field represented as a Lie algebra element.*

$$A = \sum_{i \in \#F} \boldsymbol{s}_i. \tag{4.49}$$

The simple example, because it is a 2D system with translations only, requires a 2D transformation space of reference, so $\boldsymbol{s} = (s_x, s_y)^T$ specifies a valid activation descriptor in the translation subgroup of $SE(3)$. To become a useful region of interest

detector, the accumulated activation is not computed for the complete perceptive field at once, but at higher spatio-temporal resolution instead. The accumulated activation in a spatio-temporal patch

$$A = \sum_{i \in \#F_{SD}} \boldsymbol{s}_i + \sum_{i \in \#F_{VE}} \boldsymbol{s}_i - \sum_{i \in \#F_{IOR}} \boldsymbol{s}_i \tag{4.50}$$

is furthermore influenced by bottom-up excitation from scene dynamics (SD) with positively weighted features, by top-down inhibition-of-return (IOR) with negatively weighted features, and top-down volitional excitation (VE) again with features of positive weight[9].

Typically, a detected object in a dynamic scene would be tracked by higher-level modules and further activation can be arbitrated, as the anticipated pose and motion of the object would inhibit further attention attraction on the region it refers to in the perceptive field. The effect is, that patch (1) in the FIGURE 4.11 receives high activation (in the x-dimension) in feature space, because there is only bottom-up excitation. Higher-level processes like an object detector can then react to this activation. In patch (3) which overlaps with a region containing a known object, the bottom-up excitation on the other hand gets eliminated by top-down inhibition-of-return (negative feedback) and no activation is induced on that patch.

4.2.4 Application: Tracking Interaction Partners

As a demonstration, the process described above is shown in a human-robot interaction scenario. For intuitive human-robot interaction it is essential to give the human user a feeling of presence and reaction from the robot partner. In order to fulfill this task, the robot system in the scenario detailed in SECTION 5.1.2 implements a face tracking mechanism, which is used to drive the *Philips iCat* head[10]. The effect is that the iCat head turns towards the user the robot is interacting with.

For natural interaction it also makes sense not only to recognize one single partner, but instead realize that for example a second person enters the scene. Follow-up projects to the JAST project like the EU-funded JAMES project on "Joint Action for Multimodal Embodied Social Systems"[11] focus on this aspect of human-robot interaction and foster research in this field.

[9] Sometimes bottom-up excitation from scene dynamics and inhibition-of-return are subsumed to *exogeneous* or *reflexive* attention effects, while volitional excitation is called *endogenous* attention control [CORBETTA AND SHULMAN 2002].

[10] `http://www.research.philips.com/technologies/robotics.html`

[11] `http://james-project.eu`

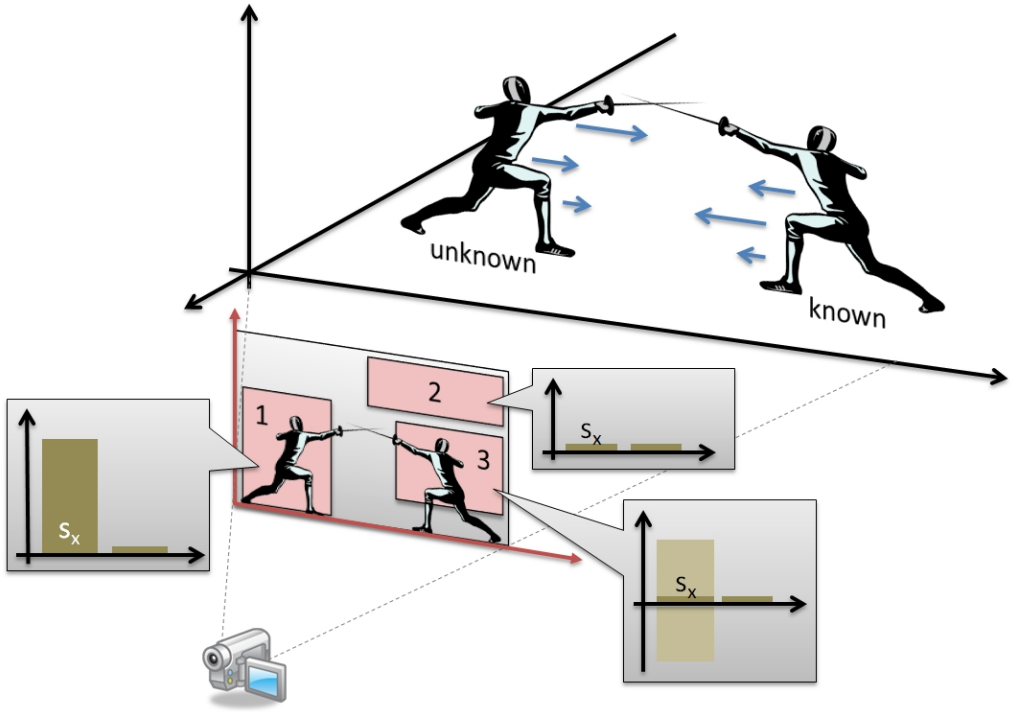

Figure 4.11 *Attention activation for different spatial patches: (1) unknown object: high bottom-up excitation, (2) no object: neither bottom-up nor top-down excitation, (3) known object: bottom-up excitation is arbitrated by top-down feedback.*

The contribution towards this direction in the interaction partner recognition domain is discussed in the following in greater detail. The approach is based on motion based bottom-up attention generation (Section 4.2.2), Haar cascades and camshift for face detection and tracking and top-down projection of high-level scene knowledge with inhibition-of-return (IOR) mechanisms (Section 4.2.3).

Bottom-Up Attraction

Uniform object motion in the perceptive field can be modeled as activation in the Lie algebra of the 2D transformation space corresponding to the image plane according to Section 4.2.2. This activation is represented using the activation features and describes bottom-up attraction of attention in a temporal and spatial context. In the scenario the temporal context of an attractive feature always corresponds to the time interval between two consecutive image frames, while the spatial context is the position and size, i.e. a region of interest, in the image plane. Figure 4.12 shows the bounding box of corresponding activation features as described in Section 4.2.2 indicated with a red outline.

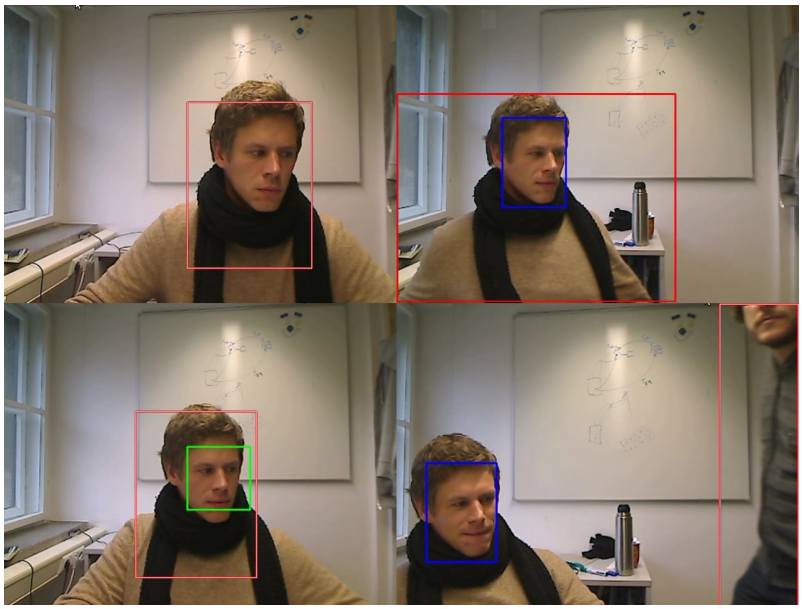

Figure 4.12 *Left-top: region of attention (white-red), left-bottom: detection result (green), right-top: initialized tracker (blue), right-bottom: tracker result (blue) and new region of attention (white-red).*

Face Detection and Tracking

Once the bounding boxes for regions with spatio-temporally corresponding activation features attract attention in the recognition system, the system tries to detect an interaction partner in the regions of interest. For this task an object detector proposed by [Viola and Jones 2001] and improved by [Lienhart and Maydt 2002] in its OpenCV implementation is used. The object detector is based on *Haar*-like features used in a cascade (i.e., a complex comprising multiple simpler classifiers). It needs a training file, which for this scenario is provided with the OpenCV library. The detection result is shown in Figure 4.12 and Figure 4.13 with a green bounding box. If a region of attention contains a face, i.e., the face detector successfully responded and a green box can be drawn, an interaction partner is detected.

Due to the time consuming face detection algorithm, it is neccessary to track the interaction partner using a realtime tracking method. For this task the camshift tracking method [Bradski 1998] is used. Camshift stands for "continuously adaptive meanshift" and is based on the meanshift algorithm [Comaniciu et al. 2000] with an additional abstraction from object size (scale) and orientation. The method evaluates color statistics, so basically it is an optimization based on color histograms.

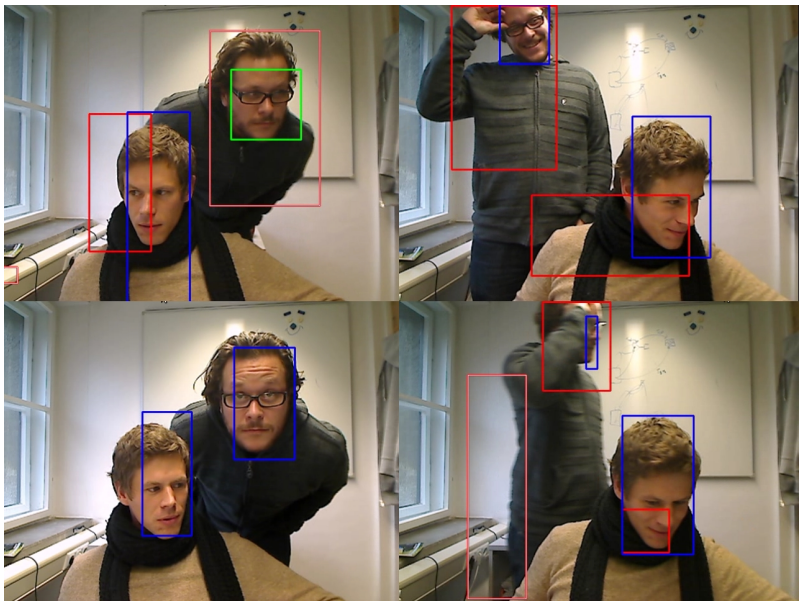

Figure 4.13 *Left-top: simultaneous tracking and detection, left-bottom: tracking without bottom-up attraction, right-top: tracking and inhibition-of-return, right-bottom: inhibited and non-inhibited regions of attraction.*

Once meanshift finds a new optimized object center using the old position in the image, camshift optimizes on the scale and orientation of the object.

In the figures the tracking result is shown with blue bounding boxes. Note, that the tracker does not need any bottom-up attraction to operate, it computes on the image directly. In this way an interaction partner can be tracked, even if no motion in the image and thus no bottom-up activation is present (see left-lower image in Figure 4.13).

Top-Down Activation Feedback

The most interesting part of the interaction partner recognition system comes next. The level of attention payed to a specific region in the perceptive field with bottom-up activation is not only based on the aforementioned bottom-up mechanism, but may also be influenced from the tracking result. The system here uses negative back-projection of the activation features present in higher-level modules through tracking. The high-level tracking result in this case is represented as an activation feature with a rectangular box as its spatial context in the 2D transformation space corresponding to the image plane, so back-projection (blue bounding box in the

figures) is only an identity transform on the track in this scenario.

The crucial part is, that bottom-up activation represented by regions of object motion in the image plane is auto-inhibited if it corresponds to back-projected activation from tracks. A spatio-temporal correspondence according to Section 3.2.1 (and as a result a negative weight on the bottom-up regions of attention) is indicated in the figure with red only color. Regions of attention that are not exposed to an inhibitory feedback on the other hand are indicated as white boxes with a red outline. For example, in the right-lower image in Figure 4.13 two red regions correspond to a tracking result, while the third bottom-up region does not.

This mechanism allows for efficient realtime interaction partner recognition, as only the non-inhibited bottom-up regions of attraction are actually passed as activation features (regions of attention) upwards to the activation association layer and in the following a cognitive module with the face detector. For the human-robot interaction scenario, a newly detected face can hence again be passed upwards into the higher-level cognitive system to react on that event. As a result, for example the robot manipulator can move its end-effector towards the new interaction partner for a handshake (details on such motor responses, direct or on high-level command are discussed in Section 4.3 in greater detail) or speech synthesis can express a "hello", etc., to confirm that the robot noticed the new person.

4.2.5 Application: Workspace Surveillance

To show the integration of an activation mechanism into a more complex robot system this section discusses the workspace surveillance system implemented as a part of the JAST human-robot interaction system (see Section 5.1.2)[12].

Basically the JAST robot needs to know which objects are present in the workspace for joint assembly (see e.g. Figure 4.15). Thus an object recognition module is developed for determining the type and precise location of any construction part on the table. As the camera device is facing downwards from the ceiling, it also seems natural to use the image information to extract gestures of the human and give feedback on the robot motion to the cognitive layers in the system.

As the different visual processing units have to work on the same image resource, the proposed software framework for parallelization (Appendix B) is perfectly eligible for this task: the system comprises a single data acquisition unit connected to the camera hardware), an early processing unit for region of interest computation, a communication unit connected to the higher-level cognitive modules of the dis-

[12]Preliminary research on workspace surveillance is discussed in [Müller 2006] and [Müller 2007].

tributed system, and several region analysis units. All of these units run in parallel and thus, it is possible to avoid waiting for a complete analysis of the workspace, but instead upward propagate (publish) the results present at any time. FIGURE 4.14 shows an overview of the units and data channels for the workspace analysis system.

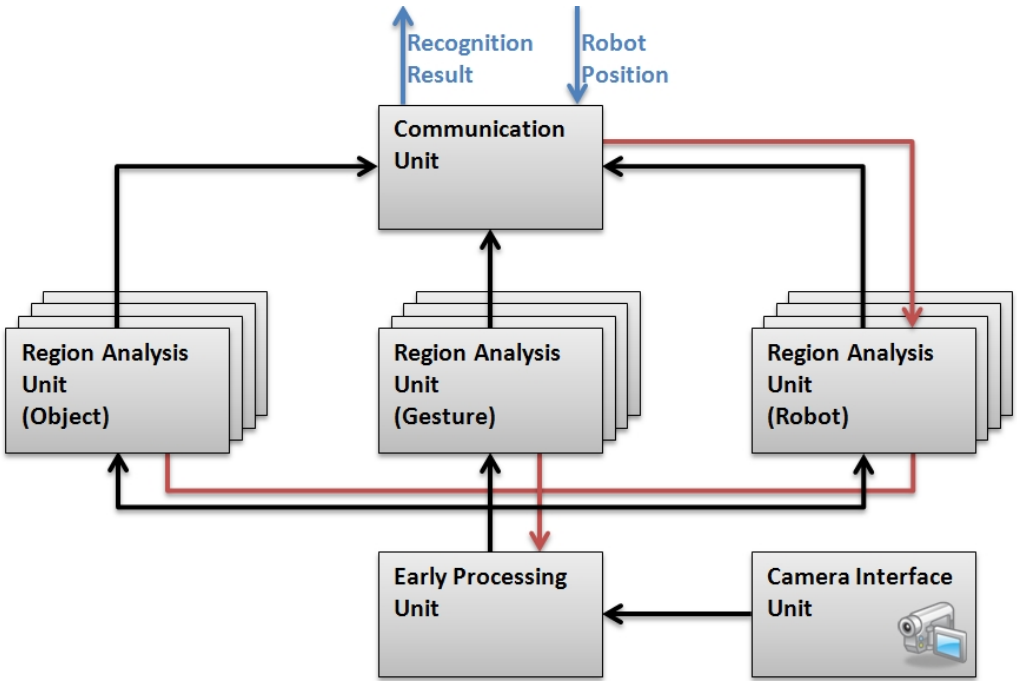

Figure 4.14 *Workspace analysis system in the JAST scenario with data channels (black and red) and IO-connections to cognitive layer (blue).*

Early Processing

The early processing unit in the JAST workspace analysis scenario implements techniques discussed before. This means the unit computes an activation in the perceptive transformation space, on a subgroup of $se(3)$ in the 2D image plane. Regarding SECTION 4.2.2 the motion detector computes bottom-up activation directly in the subgroup, i.e. in the Euclidian group $se(2)$. Thus, as soon as something enters the field of view of the camera from some direction, activation features are generated for the corresponding spatio-temporal context. The early processing unit incorporates feedback from higher-level units (red arrows in FIGURE 4.14), in particular feedback from the analysis units in order to suppress activation in certain regions of the field of view, i.e. modeling inhibition-of-return.

Concretely, if higher-level knowledge in form of an activation feature corresponding to the bottom-up activation is present in the activation layer, the features are fused and the result represents activation with feedback information integrated. Typically the top-down feedback is negative with respect to bottom-up activation, so previously attended regions of interest (probably containing objects, gestures or robot manipulator parts) which have already been analyzed do no longer cause attention and activation to be passed upwards.

Finally, only the regions of attention, i.e. those regions of interest in the field of view with a significant accumulated activation, are made available to the region analysis units for further processing. In the following these are called bottom-up regions of interest.

Robot Recognition

Robot recognition units in principle take a region of interest from early processing and compare it with projected activation received as an input from the higher-level robot control program (blue arrow in the figure). As the cognitive layer in the system knows the angle configuration of the robot manipulators, it is possible to project a (simplified) robot model into the 2D plane with respect to known calibration data. If a region of interest matches the projection, the region is marked as a "robot" region (see FIGURE 4.15) and passed back downwards into the early processing layer for inhibition of attention, as well as upwards to the communication unit for publishing.

Gesture Recognition

Gesture recognition units implement a two-step approach for gesture detection and classification. The first step evaluates, whether a region is plausible to contain a human gesture. As one can see from FIGURE 5.5, it is not possible to enter the scene from arbitrary directions in the scenario. As the workspace (the table) is perpendicular to an upright human body position, only regions connected to one of the 2D plane borders corresponding to the boundaries of the camera field of view are likely to contain a gesture.

The second step classifies the regions containing human gestures by means of a naïve Bayes classifier with distance weighting after extracting the hand contours (see FIGURE 4.16).

The implementation supports the three types of gestures shown in FIGURE 4.17. The figure also shows the classification result and additional information required in the higher level cognitive system, i.e., in case of a grasping gesture the grasp location,

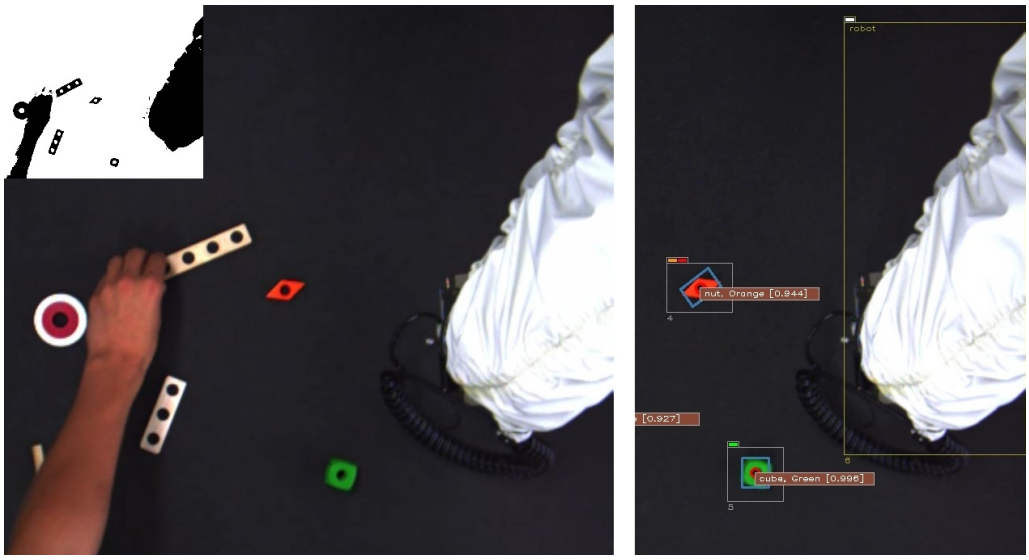

Figure 4.15 *Robot analysis compares top-down robot position feedback with bottom-up regions of interest (left-upper corner) obtained from early processing.*

in case of a pointing gesture the pointing direction, and in case of a holding out gesture the location for object hand-over.

Object Recognition

Object recognition in a region of interest passed upwards from the early processing stage is implemented as a template matching algorithm. The scenario comprises sixteen different primitive object types and four assembly objects (see FIGURE 4.18). The objects are wooden construction parts from the *Baufix*[13] domain. As the objects are of uniform color and robust for robot handling, they are ideal for a research scenario.

The object recognition system needs to identify the objects and determine their exact position and orientation for robot grasping operations. The first step in this process is analysis of the dominant colors in a region of interest, the result can be seen in the left-upper part of the bounding boxes in FIGURE 4.19.

The second step includes matching of the extracted color information with object specifications obtained from a configuration file. Corresponding object-region pairs then undergo the template matching process. Within this process several position and orientation alternatives are evaluated with respect to a similairty measure. The

[13] `http://www.baufix.de`

Figure 4.16 *Steps to find the contour used for gesture type classification: color-conversion, segmentation, and contour extraction.*

system uses the correlation coefficients measure implemented in *OpenCV*[14], which can be written as

$$R_{T,I}(x,y) = \frac{\sum_{x',y'} (T(x',y') \cdot I(x+x', y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x', y+y')^2}}, \tag{4.51}$$

where $I(x,y)$ is a pixel in the image region of interest (ROI) and $T(x,y)$ one in the template. Due to normalization bad matches refer to values close to zero and prefect matches evaluate to 1.0 respectively. By using image ROIs the system actually is able to perform the matching much faster than in a naïve approach, where the whole field of view would need be checked. Formally, x and y do thus correspond to the relative offsets inside a ROI instead of absolute camera image pixel positions.

Nevertheless, the OpenCV implementation only uses the original template orientation. Hence, to evaluate the orientation of a part on the table, the template has to be rotated and matched several times in different rotations in order to find the correct orientation of an object on the table. This process is very time-consuming and benefits greatly from parallelization and in particular from region selection by activation in the attention system of the proposed framework. Inhibitory top-down feedback on already known regions thus avoids wasting computing time on analyzed and processed objects.

4.3 Intention and Coordination

In this section two aspects are to be discussed. First, the principle of association of sensor activation with motor activation, and second the integration with or mediation from cognitive or higher-level coordinative modules is explained. Finally,

[14] `http://opencv.willowgarage.com`

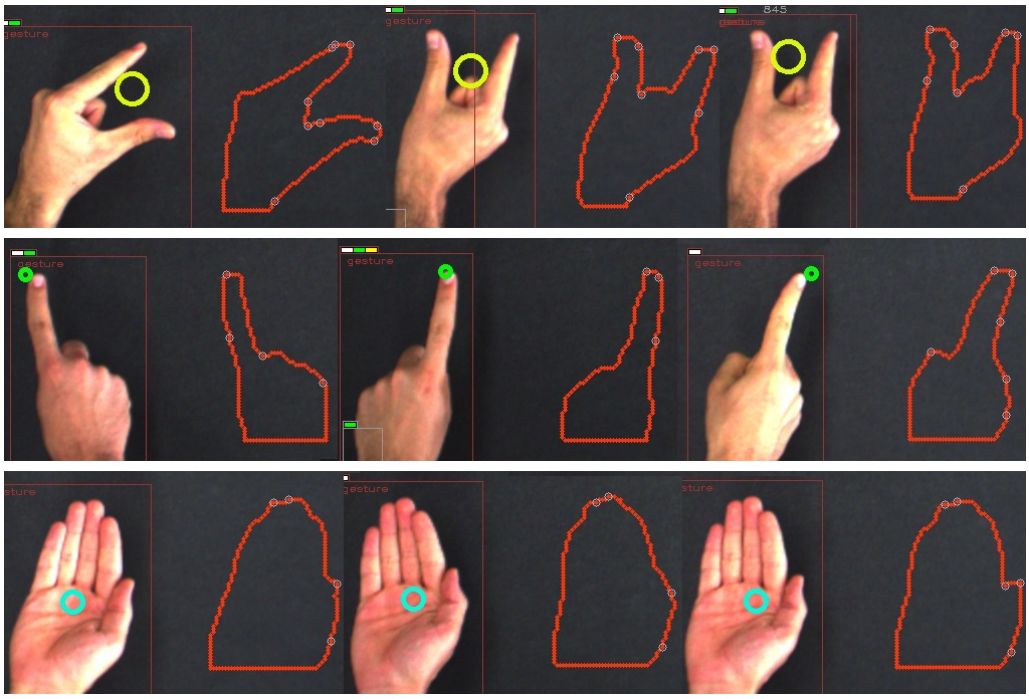

Figure 4.17 *Gesture types supported by the gesture region analysis units.*

applications in the visual servoing and active perception domain in the SFB scenario are shown.

The triplet of spatial, temporal, and excitative correspondences derived before intuitively leads to sensorimotor association and sensor-based servoing systems.[15] The approach proposed here uses the direct uplink of an activation feature, i.e., the spatial and temporal context information and the common representation in an activation reference space to introduce reflex-like motion response from the sensorimotor system on the one hand and volitionally drive the actuator(s) or introduce inhibitory effects on sensor stimuli on the other hand.

The first and almost "classical" application resembles the system of [Drummond and Cipolla 1999b, Drummond and Cipolla 2000] and comprises direct coupling of visual stimulus and motor response in a reflexive manner. The proposed system can mimic this behavior by using the Lie algebraic activation features and their correspondence mechanisms. Activation in the perceptive field is computed in order to compensate for a pose change and actuated sensor devices are thus tuned towards the desired target pose. The setup refers to a hand-in-eye configuration where the

[15] Note the relation to embodiment theory: embodiment postulates that action and perception are always associated and thus one can never occur without the other.

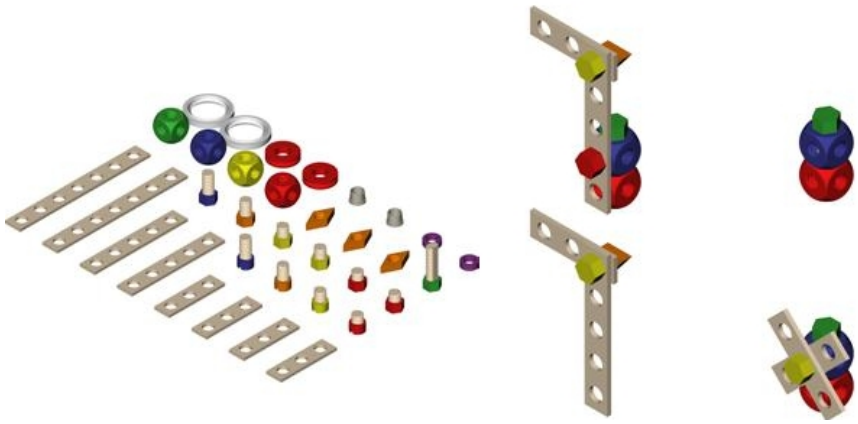

Figure 4.18 *Baufix primitive and assembly objects supported by the JAST object recognition system.*

sensor is mounted at the end-effector in an overt scenario, in a covert scenario it is almost equivalent to the attention mechanism from Section 4.2. This process is automatic as long as the association of sensor stimulus with actuator activation is concerned. The task of pose adjustment is intrinisc to this system and cannot be changed during execution – a motor response cannot be suppressed.

The second scenario integrates cognitive feedback into the first one and thus enables intentional motion and volitional inhibition of stimulus – the ingredients for deliberate and not only reflexive behavior. In this way a sensor stimulus can be mediated directly and for example instantaneous motor response can be prevented. Basically, from a higher level cognitive layer the system projects activation (or counter-activation) through the activation association layer into the perceptive field of a sensor or the motor system, which in turn causes damping or reinforcement on the Lie features there.

4.3.1 Reflexive Motion from Sensor Stimulus

The principle for reflexive motion from sensor stimuli through activation association is applied to the a problem domain consisting of a visual sensor and a 6R serial manipulator.

The term *visual servoing* then refers to the task of controlling the motion of a motor system ("servoing") with "visual" perception support (Figure 4.20). While this is the standard terminology, the approach presented here also applies to sensor based servoing in general. However, it is assumed, that the servoing system is calibrated. This means, to the system controller the position and orientation of sensors relative

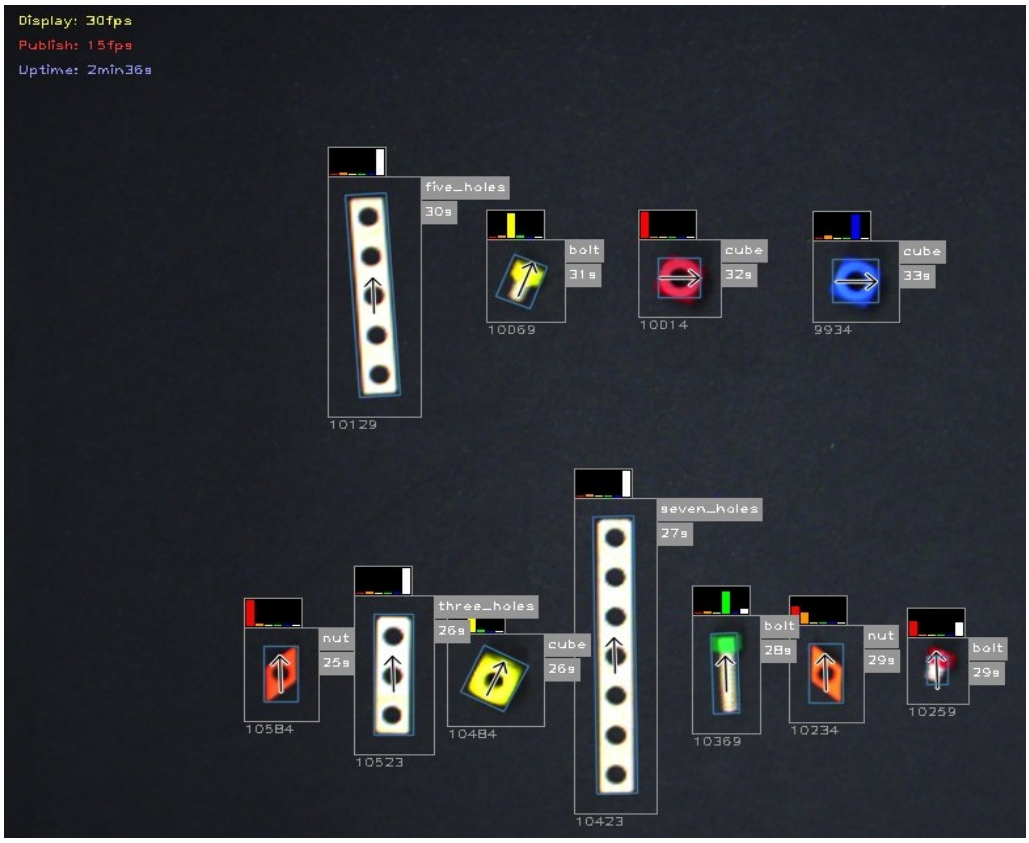

Figure 4.19 *Object recognition result: classification, position, orientation, and color information is computed in parallel and in realtime by multiple analysis units.*

to the actuator of consideration is known.

In a visual servoing application representing activation features of perception and actuation in the same reference space must always be possible. This can be the sensor space $SE(2)$ directly or the actuator space $SE(3)$, where $se(2)$ or $se(3)$ would be the Lie algebra spaces for representing activation respectively. In the vision domain in literature (e.g. [Chaumette and Hutchinson 2006, Chaumette and Hutchinson 2007]) representation in sensor space refers to image-based servoing, while representation in actuator space refers to position-based servoing. The proposed approach thus is a generalization of traditional approaches to arbitrary transformation spaces. In general, the goal is to find activation features for the actuator controller that correspond to moving it towards the desired target (see Figure 4.21).

Considering the actuator space as the common space of reference, a transformation

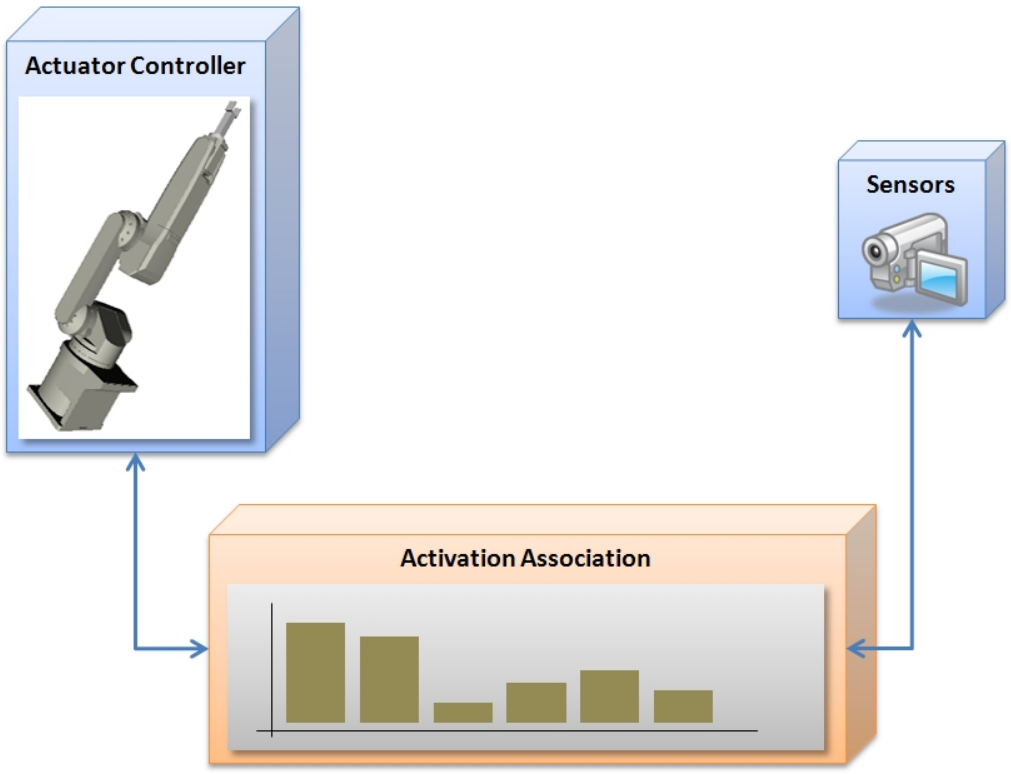

Figure 4.20 *The association layer accumulates activation and propagates from sensors into the motor system.*

moving the actuator towards the target, i.e. a tool for grasping, can be computed within n perceptive fields of a sensor set $(S_0, S_1, \ldots, S_n)$. Initially, for this purpose the actuator (the end-effector) pose is projected into the perceptive fields and the distance to the 2D pose of the target (tool) is evaluated. An activation feature F^{S_i} corresponding to the pose-to-target transformation in a sensor space S_i can then be projected into the 3D common (actuator) space by means of the feature's 2D spatial context (the projected actuator pose) and the calibration information of the sensor i

$$F^{S_i} \mapsto F_i^C. \tag{4.52}$$

An activation in the common space F^C refers to a normalized sum of input features from multiple sensors, the more the more accurate, and the actuator activation F^A

$$F^C = \frac{1}{n} \sum_{i \in n} F_i^C. \tag{4.53}$$

The temporal context is also clear: it corresponds to the actuator update rate directly. The spatial context of the features F_i^C in common space is defined up to some

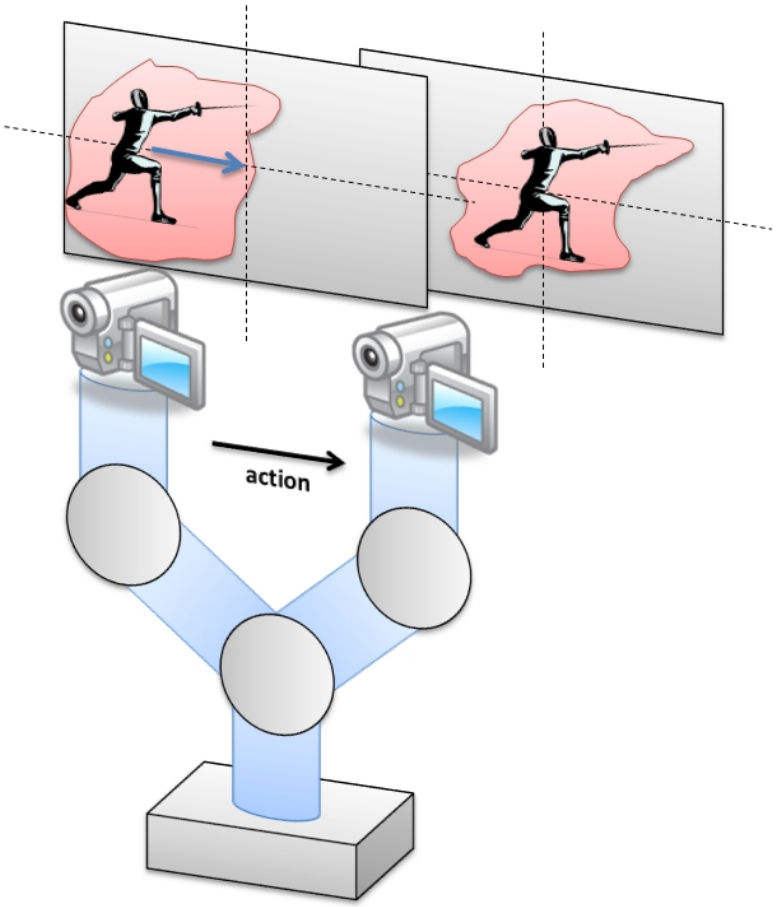

Figure 4.21 *A visual servoing setup: activation is generated from a region of interest in the perceptive field. The active perception system actuates the sensor in order to compensate for a displacement from the center of the field.*

uncertainties from the 2D-3D reconstruction as well. By means of spatio-temporal correspondence the activation is then transferred into an actuator activation F^A. As the actuator space was chosen to be the common space, the transfer is as simple as

$$F^A = F^C = \frac{1}{n} \sum_{i \in n} F_i^C, \tag{4.54}$$

or equivalently

$$F^C = \frac{1}{n+1} \left(F^A + \sum_{i \in n} F_i^C \right), \tag{4.55}$$

which is valid in the spatio-temporal patch of the end-effector's current pose [16]. This

[16] Recall, that for combining features (in particular the Lie algebraic activation), i.e. using the $\sum$ in the

works, as the spatial context for introducing an actuator (end-effector) motion is known from its current pose (see SECTION 2.3.3 for details). Hence in this application scenario the actuator activation can simply be seen as a writable feature to which the sensor activation is written. Once the activation has been passed to the action domain in this way, the next pose for the end-effector is automatically computed according to SECTION 4.1 and the corresponding motor commands are sent.

To close the loop, also F^C is mapped back into the sensor spaces

$$F^C = F_i^C \mapsto F^{S_i}, \tag{4.56}$$

which in turn specifies the next pose of the end-effector in the perceptive fields. If this still matches the pose-to-target transformation, the features F^{S_i} are propagated again and the motion commands are iteratively computed. If the target pose has changed meanwhile, F^{S_i}'s are modified accordingly. As a special case, when the target pose has been reached, the activation become zero in the sensor space, so in the end also the propagated activation feature becomes zero in the common space, as well as the actuator activation, when the target is reached.

As mentioned before, this resembles the mechanisms for visual servoing shown in literature [DRUMMOND AND CIPOLLA 2000, CHAUMETTE AND HUTCHINSON 2007], only by means of a more general methodology. The advantage is that the approach can now be extended to incorporate other modules mediating activation in the association layer. Therewith reflexive sensorimotor response can be modified voluntarily from higher levels.

4.3.2 Application: Visual Robot Guidance

The visual robot guidance application (FIGURE 4.22) follows the approach described in [DRUMMOND AND CIPOLLA 2000], except that it uses the association layer for transferring visual inputs into the manipulator domain.

The goal of this application is controlling a serial 6R robot using visual cues. The cues are evaluated from a camera image where a colored marker cross is presented. Preliminary work was done in [PLOPSKI 2010] and further development integrated the relative pose estimation from this system into the activation association mechanism.

Posing the marker in the visual field results in a transform describing the relative displacement of the marker corresponding to an initial / target pose in the image

above, special restrictions apply to $SE(3)$. For example the BCH-formula has to be used to compensate for non-commutativity or an appropriate approximation is necessary (see SECTION 3.2.2).

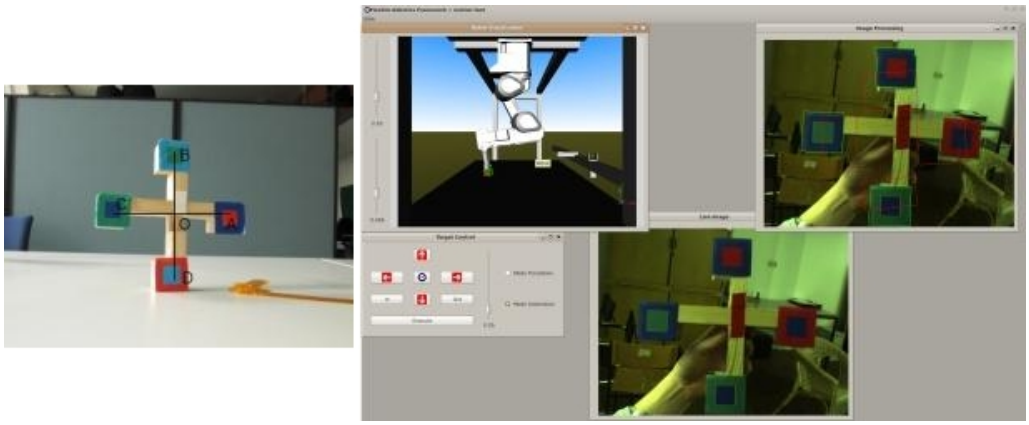

Figure 4.22 *Guiding a robot by association of activation in the visual field originating from a marker-cross with motor commands in the activation layer.*

center. The transform then undergoes a logarithmic transform in order to obtain the Lie algebraic activation values. The Lie activation is finally linked to a temporal context (i.e. the current timestamp or frame) and pushed into the association layer. The spatial context in this application is global, i.e. the robot end-effector is always associated with the activation feature.

Instantaneous motor response results from propagated activation, this means the activation is directly passed into the actuator controller, where it is translated into a pose increment on the current end-effector position and orientation. The realtime path generation system there computes a joint angle update taking smoothness and former target poses into account as described in Section 4.1.

4.3.3 Intention Extends the Reflexive System

Intention can counteract or enforce the reflexive activation, or introduce completely new activation. In principle this refers to a projection of activation from an additional source into the sensorimotor association layer from above. The architecture therefore extends to an additional module as shown in Figure 4.23.

In Section 4.3.1 in particular Equation (4.54) shows, how activation is accumulated in the association layer (orange box in the figure) and then propagated into actuator controller and sensor modules. Integrating intention into this process is relatively straight forward. Basically, activation corresponding to the intended modulation is generated in a higher-level controller and then pushed into the activation association layer exactly in the same way as inputs from the motor system and the

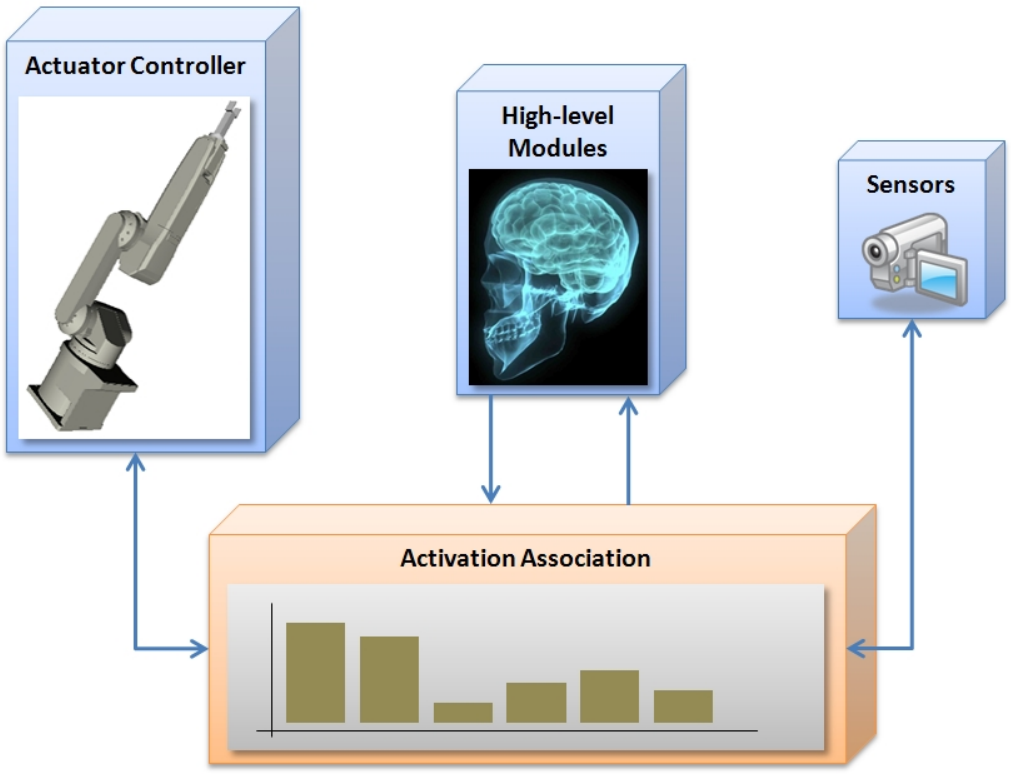

Figure 4.23 *An additional module translates higher-level decisions into the sensorimotor association layer.*

sensor system are, so

$$F^C = \frac{1}{n+1+m}\left(F^A + \sum_{i \in n} F_i^C + \sum_{j \in m} F_j^C\right). \tag{4.57}$$

Depending on the values inside the activation features this "volitional" induction into the association layer can yield different interesting effects:

- *Volitional inhibition of sensor activation*
  Projecting anticipated activation into the sensor system prevents it from generating features with significant values, so further activation and upward propagation into the higher-level module and the actuator / motor system is inhibited. The effect is a loss of the attentional focus in the cognitive modules and a suppression of a motor reflex in the actuator controller. The direct sensorimotor association loop described earlier can in this way interrupted.
- *Volitional Motion*
  Projecting positive activation into the association layer while no corresponding

sensor stimulus occurs simultaneously, yields propagation of that activation into the motor system and the sensor system. The motor system translates this activation into a motor command and the manipulator moves. The sensor system receives activation feedback and this is averaged with sensor stimulus. The sensor response is a mediated stimulus if a spatio-temporal correspondence is found – most likely the activity is lower than if the activation would not be anticipated.

The following section gives an example application for direct sensorimotor association as described in Section 4.3.1, while thereafter Section 4.3.4 shows how the effects explained above can be exploited for an attention based active operator focus scenario.

4.3.4 Application: Active Multi-Operator Focus

The idea of the application described below is the following. A "social" robot in a human-robot interaction scenario with more than one human interaction partner needs to have the capability to draw attention to interaction partners entering the scene while it must at the same time convey attention necessary for completing a task to its current counter-part. The system presented here satisfies these properties on a sensorimotor level by using the Lie algebraic activation system fed from the three modules, motor system, sensor system, and higher-level cognitive system as introduced above.

Actively focussing multiple operators or interaction partners subject to attention attractors from the environment then generally requires a system to be able to tune its perceptive devices (sensors) towards relevant aspects of the surrounding. The scenario thus implies a setup, where the sensor itself is not statically overlooking the scene, but in fact is mounted to an actuator, for example like the one shown in Figure 4.24.

In principle thus three phases in the active perception system are to be implemented in the scenario. The phases correspond to the number of human interaction partners present in the perceptive field.

In the first phase, i.e. the *detection* and initialization on attended regions phase, the bottom-up attraction introduces activation in the perceptive field (the sensor space). The attention mechanism or early information processing algorithm described in Section 4.2 condensates excitation in a specific region of the perceptive field. It generates an activity feature which can then be used as a basis for the tracker initialization, e.g. if a suitable target is found in the region.

Figure 4.24 *An active perception setup implies an actuated sensor device. In the figure, the sensor (a camera) is mounted to the end-effector of a Mitsubishi RV-6S robot.*

The active perception system in the second phase, the *stabilization* phase, always tries to optimize the pose of the end-effector (which in this case directly interacts with the preceptive field) in order to keep the region of interest in the center of the field as shown in Figure 4.21.

For this reason the sensor system computes a compensating activation feature for the target's motion. This is of course only possible if the target pose can be tracked. The compensation feature is then pushed into the association layer and propagated into the actuator system where it is translated into a motor command.

This phase also implements an ego-motion compensation algorithm. This means, whenever the actuator moves, self-induced attraction from actuator motion in the perceptive field is inhibited. The method is simple: whenever the sensor device is moved, the corresponding activation generated from static objects in the perceptive field can be anticipated, as the motor system feeds motion feedback into the sensor space. The inhibition works according to the mechanism from Section 4.2, only the sensor is mounted directly to the actuator, hence the ego-motion algorithm is straight forward. The inhibitory top-down activation F_{IOR} compensating for ego-

motion is simply the negated projection of the actual end-effector activation F_{ego}

$$-F_{ego} = F_{IOR}. \tag{4.58}$$

For example if the actuator moves the sensor leftwards, a stimulus corresponding to rightward motion is present all over the perceptive field, so this activation has to be suppressed with inhibitory (negative) top-down activation.

The third phase corresponds to *selection* or averaging of relevant features. In fact, in literature (e.g. as aforementioned in [Drummond and Cipolla 2000] or in [Chaumette and Hutchinson 2006, Chaumette and Hutchinson 2007]) one can find systems accomplishing phase two tasks directly, i.e. without the detour through an association layer and without using Lie algebraic activation features, but the advantage of the presented approach is that it generalizes on these common approaches. Therefore, it is easy to integrate other controller modules into the sensorimotor association system. This in particular appears in phase three, i.e. whenever more than one target occurs in a scene.

Consider for example the following activation equation

$$F_A = \frac{1}{n}(F_{T_1} + F_{T_2} + \ldots + F_{T_n}) = \frac{1}{n}\sum_{i \in n} F_{T_i}, \tag{4.59}$$

where the actuator system activation F_A is averaged from motion compensation features of all targets T_i in a scene on a global spatial context. The application uses an approximation to the correct form of the *Baker-Campbell-Hausdorff* formula in Equation (3.13) and Equation (3.14), which is proposed from [Fletcher et al. 2003, Govindu 2003], so averaging follows the usual vector arithmetics. This resembles the direct mechanism of phase two introduced earlier, but now extended by multiple targets and thus multiple activation features. The result is a mean compensation, which occurs as a jiggling, "indecisive" actuator motion in the worst case (e.g. if the targets move towards or apart from each other).

In the third phase therefore a higher-level module interacts with the association layer. It acts as a mediator or modifier on activation features and in this way influences the behavior of the motor system. The higher-level module may do so by emitting an "anti-"feature F_H to one or more of the target motion compensation features. In this way, some of the components in Equation (4.60) can be annihilated.

$$F_A = \frac{1}{n}\sum_{i \in n} F_{T_i} + \frac{1}{n}\sum_{i \in n} F_{H_i}, \tag{4.60}$$

where F_{H_i} either contains a zero activation vector for the features that are allowed to be propagated through or the inverse of the corresponding activation in F_{T_i} in order to inhibit the component.

4.3.5 Application: Seamless Semi-Autonomous Control

Taking the same line, another interesting application is conceivable. The idea as simple as useful. Considering an autonomous system supervised by a human operator, it is sometimes useful to be able to modify the system behavior during runtime of the system. For example the system miscalculates the target pose of the end-effector of the robot. Traditional systems mostly have an emergency shutdown mechanism for this case, so the operator has to interrupt the current process, resume a "healthy" system state and let the task continue. The methodology presented above allows for another, more elegant approach.

Once the operator detects a misbehavior, he can directly modify the manipulator motion without actually interrupting the autonomous task. This can be achieved in the association layer by adding the input activation feature generated from a manual control device. Such devices can be for example physical devices like the *3Dconnexion SpaceNavigator*[17] with six degrees of freedom, and the *NOVINT Falcon*[18] force-feedback device with three degrees of freedom as shown in FIGURE 4.25; or interaction elements (e.g. buttons) in some kind of graphical user interface.

Figure 4.25 *Supported force-feedback device with three degrees of freedom and three feeback channels and 6-dof input device.*

Analogously to the active multi-operator focus, the activation features emitted

[17] http://www.3dconnexion.com/products/spacenavigator.html
[18] http://www.novint.com/index.php/products/novintfalcon

from the manual control devices act as a high-level input module and mediate the sensor-actuator activation propagation in the association layer according to Equation (4.60).

Chapter 5

Scenarios and Discussion

Contents

AFTER elaborating on theoretical background and related work in Chapter 2, Chapter 3 derived a novel theoretical framework for describing activation in a sensorimotor robotic system based on Lie algebras. In Chapter 4 the theoretical Lie algebra activation descriptors were applied to the different domains of sensorimotor robotics, the actuator or motor system, for perception and modelling attention and finally to coordination, either for achieving purely reflexive behavior or integrating top-down intentional influences into the association system.

This chapter first introduces the research scenarios in Section 5.1 where the presented approaches are implemented, then discusses results in those scenarios in Section 5.2, and in Section 5.3 draws a conclusion and summarizes the proposed methods and techniques. Finally in Section 5.4 further experimental applications and future directions in the field of research are outlined briefly.

5.1 Research Scenarios

While some applications have already been discussed earlier in Chapter 4, this section introduces the general research context, i.e. the scenarios where these applications are useful and necessary.

5.1.1 SFB 453-T4: an Industrial Automation Scenario

Originating from a telepresence and teleaction scenario in the medical context[1] the

[1] Project SFB 453-I4 on "Shared Control for Cooperative Tele-Manipulation in Robotic Surgery: Methods, Implementation and Evaluation"

transfer towards automation in an industrial production environment is the focus of the prototypical industrial automation scenario below. This means, a sensor-assisted robotic system has to perform certain tasks autonomously, while still allowing for manual intervention with direct interaction facilities by means of a teleoperation devices and realtime workspace surveillance.

Setup and Task Definition

As a demonstration task in an experimental industrial environment a prototypical application for autonomous joining of two ends of an asymmetric rubber band was chosen. In the setup (see FIGURE 5.1) several statically mounted cameras are used: one top-view camera for workspace overview, one general camera for scenario surveillance and remote control, and another camera for visual analysis of the workpiece profile. Furthermore, an actuated camera device at the robot's end-effector is utilized for close-up inspection and accurate positioning.

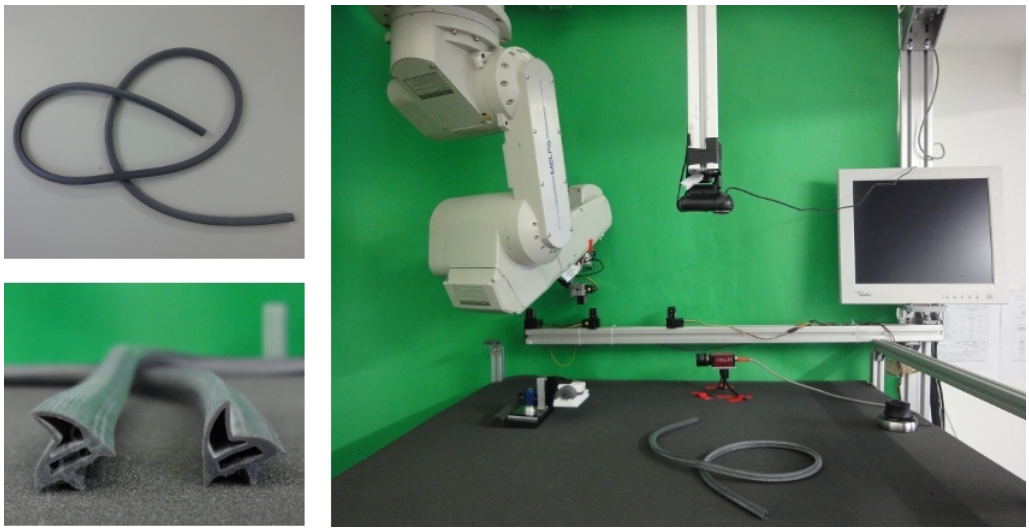

Figure 5.1 *The rubber ring joining setup for SFB 453-T4.*

On the actuator side the setup comprises a *Mitsubishi RV-6S*[2] industrial manipulator with six degrees of freedom mounted top-down, a guide plate for joining the ends of the workpiece, and several different types of electrical servo grippers (see also FIGURE 5.2), either mounted to the robot's end-effector or statically mounted to the

[2] `http://www.mitsubishi-automation.de`

gantry.

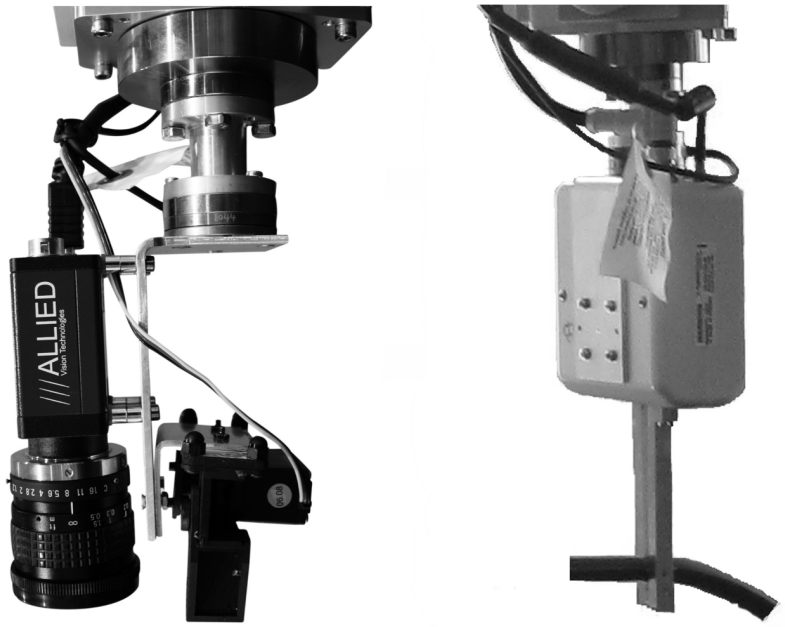

Figure 5.2 *Two types of gripper devices used in the industrial automation prototypical setup. Both devices are mounted to the end-effector of the manipulator, the arrangement on the left also integrates an eye-in-hand camera.*

Despite performing the main joining task autonomously, the control architecture has to allow for manual intervention at any time (also from a remote location via teleaction). A telecontrol unit thus interfaces with the control device (spacemouse, force-feedback device, etc.) and maintains data channels with the control system. Supported teleoperation devices are physical devices like the *3Dconnexion SpaceNavigator*[3] with six degrees of freedom, and the *NOVINT Falcon*[4] force-feedback device with three degrees of freedom (see FIGURE 4.25); or even wireless devices as presented in [PLOPSKI 2010].

Using a such device typically results in high-level feedback into the sensor-actuator association, which influences the autonomous task execution on a sensorimotor level. Moreover the system provides means for influencing the autonomous sequence via a graphical user interface – another high-level module modifying activation in the sensor-actuator association layer.

[3] `http://www.3dconnexion.com/products/spacenavigator.html`
[4] `http://www.novint.com/index.php/products/novintfalcon`

The GUI also implements data channels to the robot's position feedback and the live surveillance video unit. Here, the system computes on-the-fly compression and decompression of the surveillance stream (see details in Section B.2.2). Also, it connects to the robot's manipulator channel and other actuator channels for interaction.

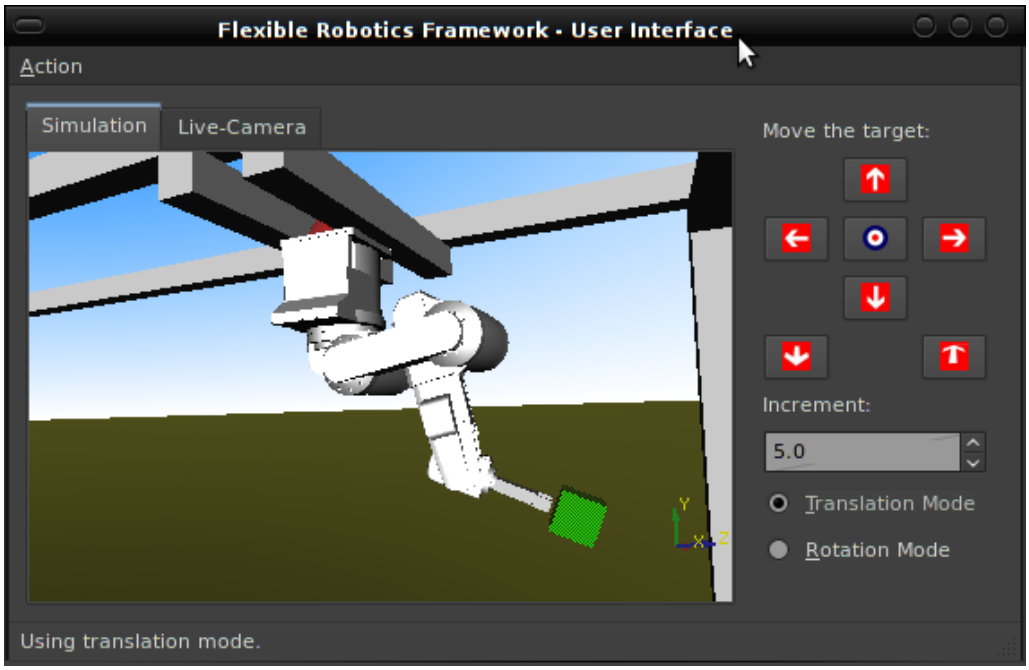

Figure 5.3 *A sample gui with live surveillance video (second tab) and a robot simulation facility. Cartesian robot target controls are also shown.*

The GUI unit comprises a *Qt*[5] window with several customizable widgets. The elements of the GUI are not per se fixed by the software framework, only the application defines the concrete widgets. Widget prototypes predefined by the framework or user defined interaction widgets can be connected to arbitrary data channels to perform visualization (e.g., video display) and realize control facilities (e.g. push buttons). For a detailed description of the software architecture please refer to Appendix B. A simple GUI built with such predefined prototypes is shown in Figure 5.3. An online video[6] shows the remote control and interaction facilities of the industrial automation scenario in greater detail.

An analysis of the demonstration task leads to the requirements described below.

[5] `http://qt.nokia.com`
[6] `http://www.youtube.com/watch?v=kO0k5cBkkdA`

1. Basically, the task needs to be implemented with a sensor-based system, as the position of the workpiece, a limp rubber band, is not fully predictable during task execution, but instead has to be monitored using sensor devices.
2. Furthermore, realtime scenario surveillance for remote control of the system needs to be implemented. It shall in this way be possible to recover the system manually from an error state, which eventually involves interrupting or correcting the current autonomous task and overtaking control using a teleoperation device. This also implies it must be possible to view a live video stream of the setup during task execution for monitoring.
3. Considering the workpiece, its structure needs to be evaluated in order to determine possible grasping points. Also, hardware and software means have to be provided to avoid unwanted twisting before the join of the ends. A higher-level autonomous error recovery strategy has to be implemented in case the workpiece has insolvable knots or unexpected problems occur during execution.

The software architecture presented in Appendix B is used to implement the system. The application task is decomposed and decoupled into sub-tasks and thus several processing units can be implemented to fulfill the desired requirements in parallel. Interaction, i.e., data exchange between the units and event generation defines the application workflow.

Robot Puppeteer Extension

A slightly modified setup is the basis for another interesting experiment in this scenario. The goal of this experiment is to improve a secondary humanoid robot's gait system with reinforcement learning strategies. Figure 5.4 shows the experimental setup.

As it would be time consuming to put the humanoid (puppet) into its original position after an erroneous try, which happens very often during the gait improvement phase, this process is automated using a puppeteer manipulator. The humanoid is equipped with a force sensor to detect if it has fallen over. Once this is detected the puppeteer robot sets the humanoid into a start position again. Then a trajectory is sent to the manipulator in order to guide the humanoid along its next try. In this way it is possible to teach even complex trajectories like walking a curve or walking up or down a tilted plane at different velocities. A reinforcement strategy in the gait controller of the humanoid computes a new set of gait parameters after each episode, which incrementally result in a better gait behavior. Details on the results of the experiment can be reviewed from [Völk 2010].

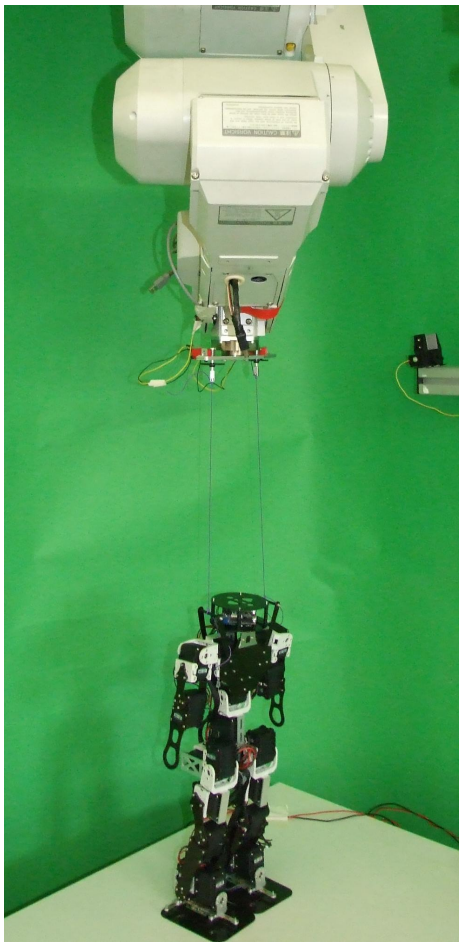

Figure 5.4 *Puppeteer setup in the scenario. The industrial robot and its modified remote control interface is exploited to play the role of a puppeteer for improving a humanoid robot's gait system.*

For the experiment the remote control unit is modified in order to be scriptable by a remote client computers via a telnet server unit. The gait controller of the humanoid robot is enabled to send Cartesian target commands. Also, Cartesian position feedback channels and a data channel with information on Euclidian distance to the current target are integrated in the manipulator server unit. The server unit then emits necessary `Success` and `Error` events and handles the communication with the task control unit on the one hand and with the telnet client on the other hand.

5.1.2 JAST: A Human-Robot Interaction Scenario

Opposing the industrial automation scenario, the purpose of the scenario introduced in the following is purely research. The topic of interest is dialog-based intuitive human-robot interaction and joint action. Fulfilling an assembly task in cooperation with a humanoid robot is the overall goal of the scenario.

In particular this requires the system to be responsive to the human interaction partner in terms of action and speech (see Section 4.2.4), and be aware of the processes in the common workspace environment by means of recognizing assembly objects, human gestures, and its own manipulators (see Section 4.2.5).

The JAST setup is based on a distributed system, which is implemented with *ZeroC's Ice*[7] platform as a middleware [Foster et al. 2006, Rickert et al. 2007, Giuliani 2011]. The system comprises two industrial robot manipulators with grippers mounted side-ways on a gantry to mimic a human upper torso. The *Mitsubishi RV-6SL*[8] manipulators are complemented with a *Philips iCat*[9], an animatronic talking head in the shape of a yellow cat (see Figure 5.5). The iCat can express facial motions and turn in two degrees of freedom, and apart from moving the manipulators in six degrees of freedom the system includes synthesized speech. The input channels consist of speech recognition, object recognition, gesture recognition, robot sensors, and face tracking. On the cognition side, the system integrates a goal inference system based on neural fields with a classical rule-based assembly planning paradigm [Foster et al. 2008].

The components developed with the proposed methodology and software framework focus on visual perception and attention based early processing. This includes workspace analysis, i.e., observing the table with a top-view camera with respect to robot parts, gestures, and wooden assembly objects (Section 4.2.5); and environment surveillance and interaction partner recognition, i.e., detecting faces of interacting users with a camera device (Section 4.2.4) based on attention attractors.

As mentioned above, the JAST setup is based on a proprietary middleware, *ZeroC's Ice*. A limitation of this of course is, that the processing modules of the overall system are not distributed at runtime, but instead have to be configured in

[7] http://www.zeroc.com
[8] http://www.mitsubishi-automation.de
[9] http://www.research.philips.com/technologies/robotics.html

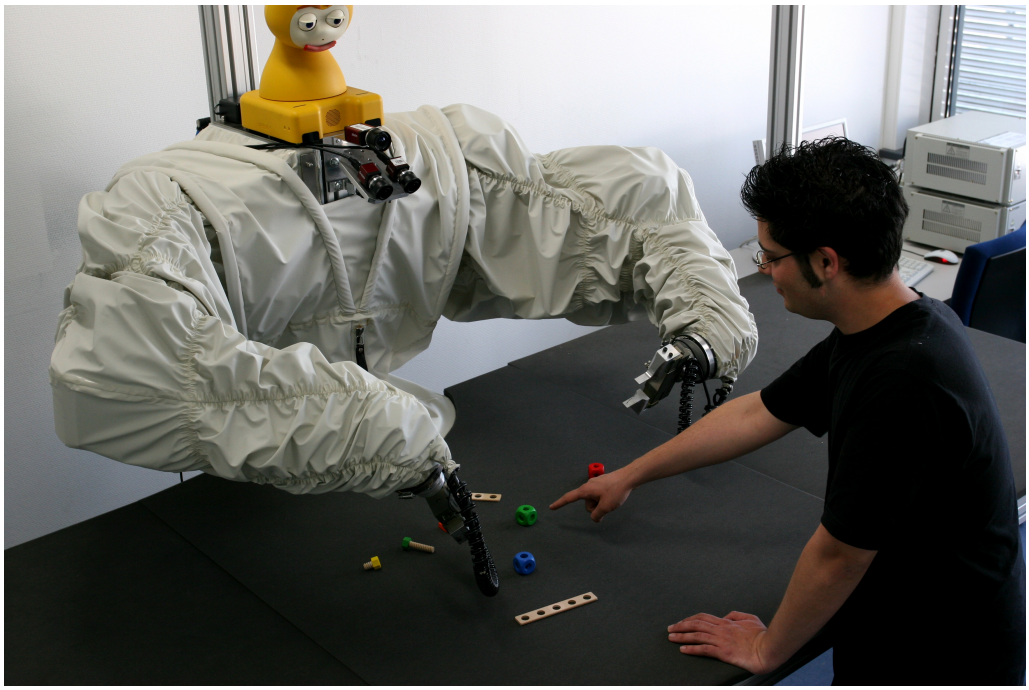

Figure 5.5 *The human-robot interaction setup for JAST.*

advance, i.e., the computers that run Ice components have to be specifically configured to run a certain executable, that in turn establishes a connection to other distributed resources. Nevertheless, the scenario demonstrates integration of a possibly distributed subsystem based on the software framework from Appendix B with external software components. The two example modules shown, i.e., one for object recognition and realtime workspace observation and one for interaction partner detection and tracking, seamlessly interact with the external Ice-based middleware using dedicated communication units. These communication units realize and encapsulate the necessary Ice interfaces and handle all data exchange with the other components, so the task processing units in the presented subsystems do not need to explicitly take care of communication issues.

5.1.3 Further Scenarios

Apart from the industrial automation scenario and the human-robot interaction scenario other scenarios with respect to the work of thesis can be identified in the automotive field. Preliminary research in this field has already been carried out in

cooperation with *Audi Electronics Venture GmbH*[10] and *fortiss GmbH*[11].

The scenario on the one hand comprises activation correspondence of GPS, environment model and visual perception data for navigation and localization. An early system tried to align both activation generators and find a better approximation of the vehicle position. Figure 5.6 shows the aligned environment model data projected into the image plane.

Figure 5.6 *Fusion of GPS data and visual perception for navigation and self localization.*

On the other hand autonomous obstacle avoidance and pedestrian recognition are an interesting field of application for methods proposed within this work. A first implementation tried to apply the template matching approach from Section 4.2.5 on a depth image to the pedestrian recognition domain (see Figure 5.7). The result was not satisfying, as the system detected too many false positives.

A more sophisticated attempt uses histograms of oriented gradients [Dalal and Triggs 2005] for human detection in a stereo camera setup and a particle filter for tracking the human(s) in the scene. Direct linear transform is used for 3D reconstruction on corresponding points in both views retrieved with SURF [Bay et al. 2008]. The tracker is then updated from time to time with a detection result for increased robustness.

[10] `http://www.audi-electronics-venture.de`
[11] `http://www.fortiss.org`

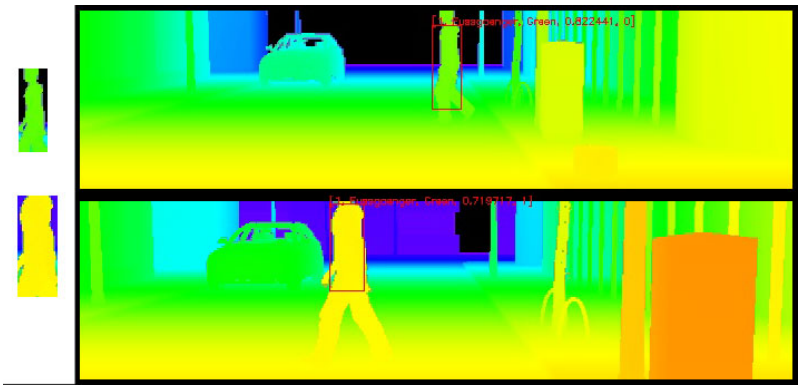

Figure 5.7 *Template matching for pedestrian detection.*

The result (see FIGURE 5.8) is promising, as shown in [TUONG ET AL. 2011] and [TUONG 2010]. Still, the approach should be more successful using the attention mechanism with inhibition-of-return based on knowledge of ego-motion and its back-projection into the common space of transformation as presented in SECTION 4.2.

5.2 Results and Discussion

This section elaborates on the results achieved for the research scenarios above. It discusses solutions and approaches provided by the work in this thesis and shows limitations and further potentials.

5.2.1 Manipulator Control

The SFB scenario requires a multi-channel input to the actuator control system. This means, multiple input modules need to be able to modify the Cartesian end-effector target pose at the same time. Therefore, the manipulator controller must be able to recompute a trajectory in realtime according to a possibly modified target pose. Furthermore, the path generator must generate smooth paths in order to prevent the manipulator from mechanical damage due to abrupt motion commands.

As proposed in SECTION 4.1, the Lie algebraic activation feature system provides exactly these properties. The realtime target pose update is implemented such that the pose is only incremented, but never set to an absolute value directly, as the increment is computed from the exponential map on the value vector in the Lie algebra corresponding to the transformation for the increment.

The algorithm applied is derived from a method for moving towards a static target

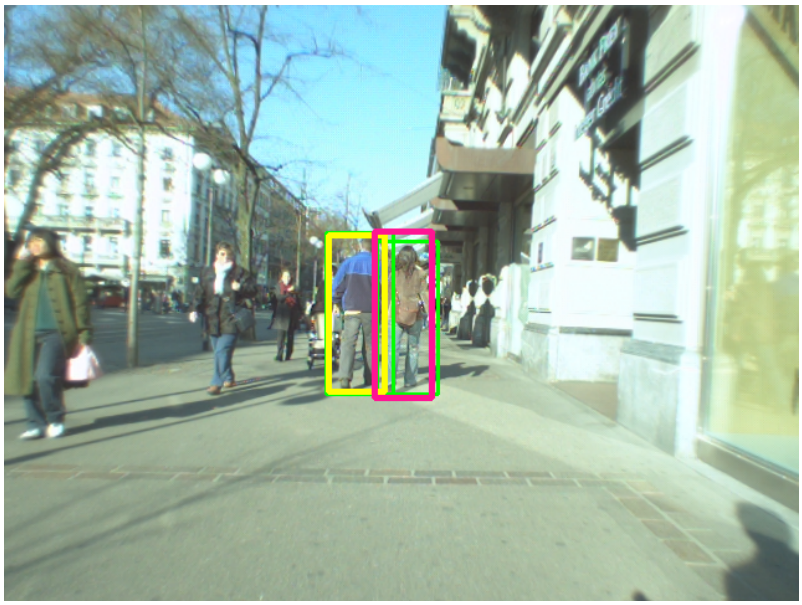

Figure 5.8 *Particle filter based pedestrian tracking result with histogram of oriented gradients detector feedback (yellow/pink: tracker result, green: ground-truth).*

pose according to the manipulator update rate. With this method (Section 4.1.1) the end-effector path can be configured to comply with different velocity profiles, be it constant, trapezoidal, or S-ramped profiles. This method is extended to allow for realtime target pose updates during the path generation, while the smoothness (with respect to higher order terms) of the path can be maintained (Section 4.1.2). A worst case analysis shows, that for a constant velocity profile this means the sign of the velocity vector may flip, and the position increment absolute remains constant. The same applies for higher order profiles, so example for trapezoidal velocity profiles the absolute of the velocity increment or decrement remains constant, and so on. The drawback with the online target pose update is of course, that a minimum jerk profile is in general not possible, but substituted with an S-ramped profile, because the duration of the motion is not clear from the the beginning.

The advantage of the approach is clearly, that at any time an input from the association layer can be fed into the motor controller and still the system remains in a consistent state, i.e., the motion is slowed or direction is changed smoothly.

5.2.2 Attention in a Perception System

Due to the great computational cost of visual processing tasks, it is often necessary to speed up vision systems in order to enable realtime performance. The approach presented in Section 4.2 takes advantage of Lie algebras to achieve this goal. The basic idea is to implement strategies similar to the human attention system. The algorithm is based on an attention condensation mechanism (Section 4.2.1), which has been discussed in several preliminary publications [Müller and Knoll 2008a, Müller and Knoll 2008b, Müller and Knoll 2009a], and in a much restricted scenario but with integration of evolutionary strategies in [Müller and Knoll 2011].

In this thesis this approach is improved and integrated into the Lie algebraic framework. In particular, in Section 4.2.2 bottom-up stimulus in the perceptive field, the attention attractors from environmental influences, is represented by means of Lie activation features. In Section 4.2.3 then top-down (or higher-level) feedback is combined with the bottom-up attraction in the Lie domain, so regions of interest can be extracted for further analysis efficiently.

These methods are applied to the interaction partner recognition and tracking systems in the JAST research scenario (Section 4.2.4) and to the workspace surveillance systems in both the JAST and the SFB scenario (Section 4.2.5). The importance of fast reaction times on human-robot interaction (e.g. with respect to safety) is discussed in great detail in [Giuliani et al. 2010], and preliminary research in this field is also shown in [Müller, Lenz, Barner and Knoll 2008].

The benefit of improving performance in comparison to naïve visual processing methods can be measured in the workspace analysis and the interaction partner recognition and is easily calculated. The percentage of coverage of regions of activation with respect to the image plane directly decreases the computation times. For example, if the regions of activation cover one fifth of the perceptive field, only one fifth has to be analyzed within higher-level units. As regions of activation can also be narrowed down from regions of bottom-up attraction by inhibition-of-return, i.e., top-down effects, these effects also directly influence the computation times.

For example in the interaction partner recognition system, most of the time uniform motion is detected inside regions that already contain a recognized interaction partner. Thus, these regions can be entirely discarded from face detection and only the much faster track update has to be computed. The detection to tracking computation time ratio is about 12.7 in the experimental setup (Intel Core2 Quad CPU at 2.5GHz $\times$ 4 with 3.2 GB of memory) using a cascaded Haar classifier for detection

and a camshift algorithm for tracking. I.e., a detection cycle costs **more than ten times** compared to a track update cycle.

Furthermore, the scenario implements parallelization for multi-core machines (Appendix B and [Müller, Ziaie and Knoll 2008, Müller and Knoll 2010, Müller et al. 2010]), which scales almost linearily and thus again improves the performance of the overall system. In the workspace observation system this is due to the complex analysis of regions. Thus, the larger the number of processing units for analysis, the better the distribution on multi-core machines. For example, considering object recognition[12], the detection and classification algorithm needs to be applied to each region (usually about ten in the JAST scenario), within a region each of the twenty objects needs to be matched in several rotations (twelve alternatives on average). This results in $10 \cdot 20 \cdot 12 = 2400$ times applying the cross-correlation from Equation (4.51) per frame. This truly is a time-consuming process. Thus, if each of the regions can be processed in parallel, and thus be distributed to all available cores, this reveals great benefit to the system performance.

All together, by application of early-processing strategies and paralellization mechanisms the performance of the workspace analysis system can be improved from analysis of a single frame of about sixty seconds on average (depending on the number of objects on the table) to visual realtime (i.e. 25 Hz). In the interaction partner recognition system the activation correspondence and tracking strategy and parallelization also results in realtime performance (where the CPU load is about 35%[13]) using the experimental system compared to about 1.8 seconds (ca. 0.55 Hz) processing each frame with the face detector at full load on only a single CPU. If one estimates the performance for a high-speed camera device, where the CPU potential can be utilized completely, the improvement is about: 35% CPU = 25 Hz, so 100% CPU = 71 Hz; and this results in ca. **130 times** the performance, while the quality can be maintained.

5.2.3 Sensorimotor Association and Intention

The motivation for the approach for sensor-to-actuator propagation, i.e. closing the action-perception loop, presented in this thesis was implementing a system allowing for direct motor control from sensor stimulus in a reflexive manner on the one hand, and seamlessly manipulating the reflex-like behavior with higher-level input on the

[12] Details on the gesture recognition subsystem can be reviewed from [Ziaie, Müller, Foster and Knoll 2008, Ziaie et al. 2009], and [Ziaie, Müller and Knoll 2008], where it is stated that the system reveals about 96% correct classifications in the JAST scenario.

[13] This is due to the maximum framerate of the camera device of 25 Hz.

other hand. The approach proposed here consists of establishing a sensorimotor association layer (Section 3.1), where activation features (Section 3.2) are used for combination of inputs from different sources and back-propagation based on Lie algebraic structures is performed (Section 3.3).

Above it was discussed, how the algebraic structure, i.e. the activation feature can be utilized for interesting tasks in the manipulator and the sensor domain. Now, the association layer is discussed briefly with respect to the results achieved in the research scenarios.

As the association layer is intended to generalize common sensor-based manipulator control systems, it must be capable of resembling this behavior. The common term for these robotic setups is visual (or sensor-based) servoing systems. The standard approach translates visual input into motor commands such that the manipulator target pose corresponds to a predefined goal state. This can for example be the workpiece positioned in the view center. Achieving this goal is performed by incremental pose updates until the goal state is reached. Another example is moving the manipulator by visual displacement of a controlling device. The methodology is analogous. In both examples the displacement is continuously translated into an increment on the manipulator pose. This process is implemented on basis of Lie algebras in the proposed work (Section 4.3.1). This means, the displacement increment is not expressed in the transformation space $SE(3)$ directly, but mapped to the corresponding Lie algebra $se(3)$.

The advantage is, that in the Lie algebra, as it represents an *increment*, not an *absolute* value, calculations become more efficient due to the differential simplification. For example, normally, if a pose represented as a transformations needs to be modified several times this requires sequential matrix multiplications (details in Section A.2) and the result is not consistent, as it is highly dependent on the current pose. If the sequence is transferred into the Lie algebra domain instead, the result is consistent, as it is performed independent from the current pose – it is only applied to the current configuration after performing the computation! Therefore, round off errors minimize and computation becomes simpler, as it is done on a six-vector instead of a 4×4 matrix. Most important, now computation can be done on an abstract level, i.e. in the association layer, without actually considering the current pose of the robotic system, only after propagating the result into the motor system it is translated into the concrete pose transformation update.

The experimental visual robot guidance application in the SFB scenario described in Section 4.3.2 implements this abstract propagation of sensory input, in this case

visual stimulus, into motor commands. This resembles the behavior described in literature (e.g. [Drummond and Cipolla 2000, Chaumette and Hutchinson 2007]), while it uses the association layer instead of directly pushing motor commands into the manipulator system. Of course, the same well-known problems may occur. This includes for example inaccuracies due to faulty displacement estimations of the visual controller and thus incorrect corresponding Lie increments. Nevertheless, this can be damped efficiently with the smooth realtime manipulator control system.

Here, the advantage of the proposed abstraction towards an activation association layer becomes even clearer, if one considers the inputs of higher-level modules to the sensorimotor system (Section 4.3.3). Using the feature combination facility of the layer with the *BCH*-approximation derived from e.g. [Fletcher et al. 2003, Govindu 2003] integrating with other inputs becomes an easy task. One application of this procedure is shown in Section 4.3.4. It refers to an active end-effector orientation and pose update with respect to turning towards multiple human interaction partners based on the level of attention payed to each of them. Here, a higher-level module mediates the bottom-up sensor-based activation features and pushes "anti"-features into the association layer which enables inhibition of selected features by averaging. The application is intended for the JAST research scenario (or its follow ups like e.g. JAMES[14]), where multiple interaction partners are often present in the scene and such selection is necessary.

The feature push mechanism from multiple sources is also used for an application in the SFB scenario (Section 4.3.5). Here, in the semi-autonomous task execution process sometimes the human supervisor must be able modify a task at runtime. Therefore, controller devices (either physical devices like a spacemouse or the graphical user interface) allow for pushing inputs into the association layer. In this way, the behavior of the manipulator end-effector can be modified during task execution and thus the task needs not be interrupted, but can be continued integrating the online-adjustments seamlessly.

5.3 Conclusion and Summary

This thesis presents original work on applying continuous transformation groups and their tangential vector spaces at the identity group element, the Lie algebras, to sensor-based robotic systems. For such applications a descriptor structure based on Lie algebras of various transformation groups, most prominently $SE(2)$ and $SE(3)$, is derived. Means for generating, combining, and evaluating the difference of these

[14] `http://james-project.eu`

descriptors are discussed on a mathematical level.

Then, this work proposes applications of the Lie algebraic activation features to different domains of sensor-based robotic systems. This includes modules for efficient realtime manipulator control, sensor data (pre-)processing, and sensorimotor association and coordination.

- In the manipulator control domain, a realtime path generation system for freely moving Cartesian targets is derived. It allows for receiving target pose updates from activation features at any, while still generating smooth end-effector paths according to the hardware specifications and motion constraints.
- Thereafter applying the activation features to early processing in the perception domain is presented. Here, using the Lie algebraic constructs an attention mechanism is implemented, which includes bottom-up excitation and top-down feedback, and is similar to the human attention system. It is shown, how this attention based vision system can be used to dramatically improve performance on workspace surveillance on the one hand and multiple interaction partner recognition on the other.
- Finally, in the coordination domain the activation features can be used to realize direct sensorimotor association as necessary for traditional visual servoing applications. In addition to that, the activation features allow for seamless integration of intentional, and therefore high-level cognitive inputs to the reflexive servoing system on a sensorimotor association level. Unifying the representation of actuator-, sensor-, and cognitive activation with the Lie algebraic features, and multi-directional combination and propagation by an association layer facilitate this capability.

A short overview of all contributions with respective sections is given in SUMMARY 5.1 below.

5.4 More Applications and Future Directions

Two possible interesting applications in the field of sensorimotor robotics are introduced briefly in the following. First, transferring the insights of path generation into an application for limp structure recognition is implemented in a preliminary application, and second, auto-calibration of sensor-actuator systems by observation over time – just as humans do it in their early days of live, would be great applications.

Furthermore, in the theoretical field of group theory with respect to sensorimotor

Summary 5.1 – **Contribution Overview**

Theoretical derivation of Lie algebraic activation features:

- Lie theory and algebras – SECTION 2.2 and SECTION 2.3
- Feature structure and generation – SECTION 3.2 and SECTION 3.3

Lie descriptors in the actuator domain:

- Activation based realtime path generation – SECTION 4.1.1
- Smooth realtime paths for freely moving targets – SECTION 4.1.2
- Implementing a control system – SECTION 4.1.3

Lie descriptors in the perception domain:

- Activation basics in the sensor space – SECTION 4.2.1
- Top-down and bottom-up activity – SECTION 4.2.2 and SECTION 4.2.3
- Improving vision systems – SECTION 4.2.4 and SECTION 4.2.5

Lie descriptors for coordinative tasks:

- Reflexive behavior – SECTION 4.3.1 and SECTION 4.3.2
- High-level intentional inputs – SECTION 4.3.3 to SECTION 4.3.5

robotics one more interesting property can be investigated in greater detail in the future: dual quaternions as an elegant representation of transformations in $SE(3)$, as well as their orthonormal basis. Probably this representation has the capability to further simplify the combination and averaging methodology in the activation feature space.

5.4.1 Feature-based Limp Object Recognition

The principle idea of the approach presented in the next sections is to apply the continuous path generation method from the last chapter to the domain of linear object detection. In general, the idea is based on a simple thought: as little as a path generated for a robot end-effector is allowed to make abrupt direction changes, also the direction when "walking" along a deformable linear structure (such as a cable) is likely to change within an infinitesimal increment (see FIGURE 5.9). Both

is due to physical constraints, i.e., in the case of a robot path mass inertia applies and the consequence would be e.g. damage to the robot, while in the linear object domain other physical constraints apply (for example material stiffness) and the consequence would be e.g. cable break. So it should be possible to apply the smoothness constraints of robot end-effector paths to the detection algorithm for limp structures in the visual perception system.

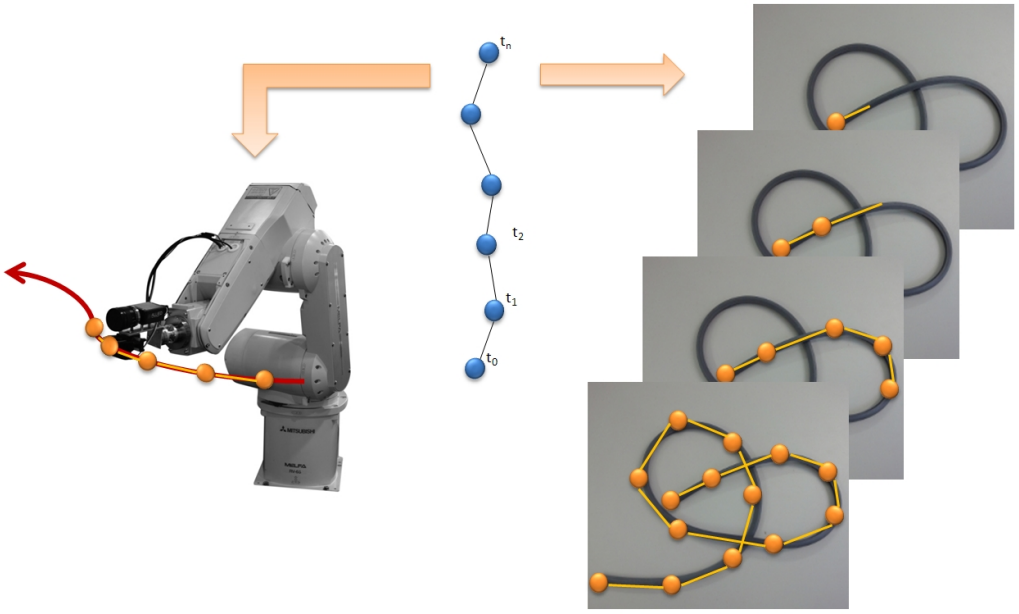

Figure 5.9 *Analogy for transferring from the robot (action) domain to the vision (perception) domain.*

Modeling Limp Objects

A model for linear deformable structures can be built online using the path generator in the visual domain. Few other approaches have been proposed for this task, all of them lacking sufficient robustness or efficiency for realtime visual processing due to the large number of degrees of freedom. Hence, reducing the scope of the problem by adequate simplification, e.g., with chain models, mass-spring systems, or finite-elements methods (FEM), is essential. A review on chain models can be found in [Joukhadar and Laugier 1997]; an overview and limitations of mass-spring systems is provided for example by [Loock and Schömer 2001] or [Provot 1995].

As the continuous robot path is discretized technically with the robot update interval, consequently also a discretized model for the continuous linear object is computed. A such model is usually called a chain model and comprises nodes and vertices (also *links*). While the links of the chain are simply modeled as piecewise

stiff components (see FIGURE 5.10), the nodes can be modelled as poses with a single angular degree of freedom (like revolute joints) in $SE(2)$, or with two angular degrees of freedom (like ball and socket joints) in $SE(3)$ [GRASSIA 1998]. The compositional generator is then used to iteratively interpolate a path between such nodes for retrieval of the object model.

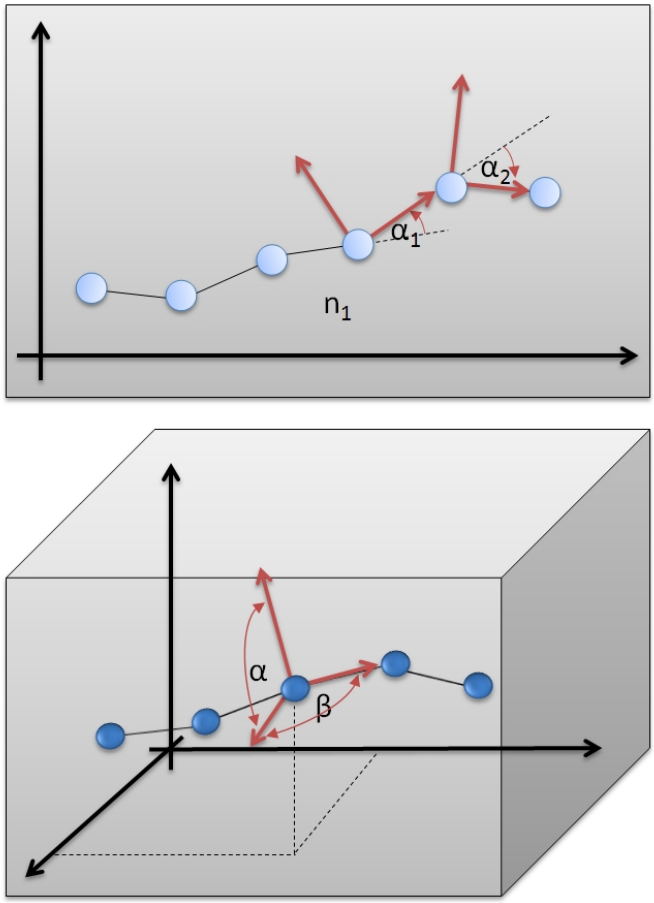

Figure 5.10 *Chain models in 2D and 3D space.*

The velocity profile used within the generator is of minor importance for this application. In fact it makes sense to use the most simple constant velocity profile, as this corresponds to a constant internode distance when the time domain in the path generation process is transferred to the spatial recognition process in the visual domain. The link length is thus represented by the velocity constraint of the path generation, i.e., the larger the maximum velocity $v_{max} = v$ in the constant velocity profile, the greater the node distance. This of course is only a "virtual" velocity, as the computation of the path is not depending on any timing or realtime constraint, but should run as fast as possible. Together with the "virtual" update interval u, it

is only needed to specify the spatial resolution $r = v \cdot u$ corresponding to the number of nodes per length unit and thus the accuracy of the generated chain model.

Recognition Algorithm

The detection, online model generation algorithm for deformable linear structures comprises several steps (FIGURE 5.11 shows a flow diagram). In general it is an iterative coarse to fine procedure with plausibility measurement and few input parameters.

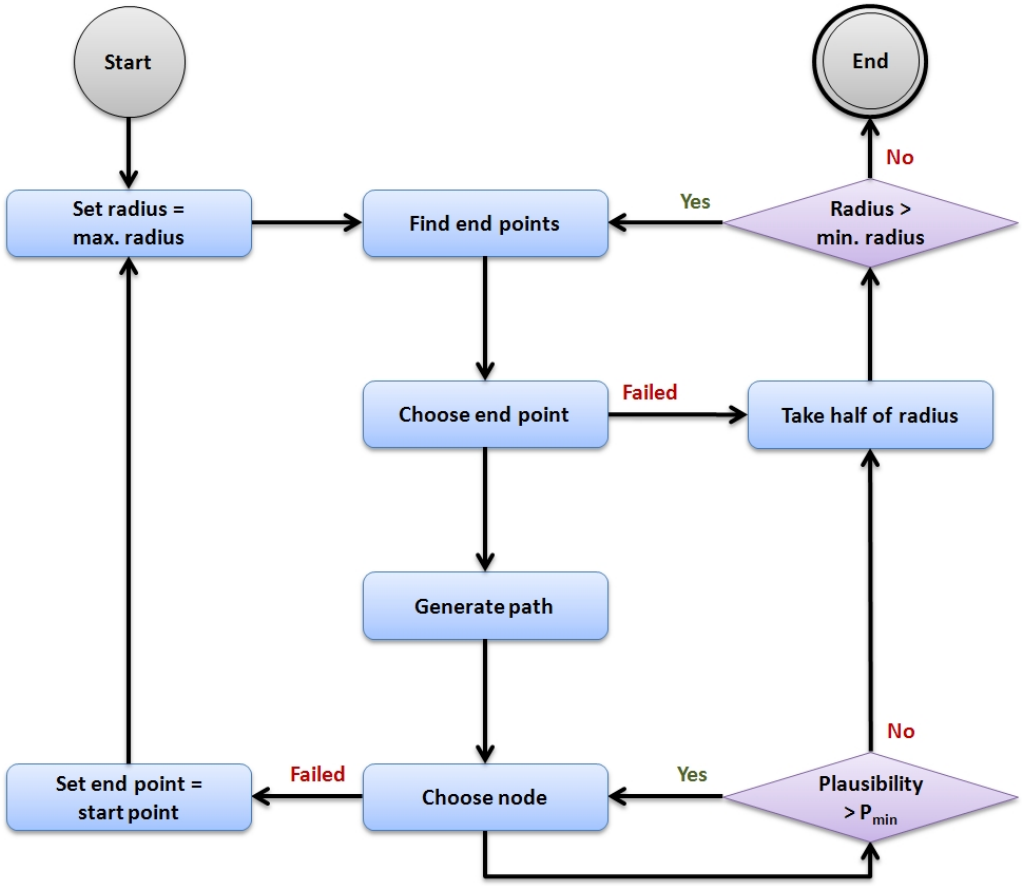

Figure 5.11 *Flow diagram for limp object recognition using the compositional path generation system.*

In advance, material properties in terms of the path generation method have to be specified. These are the velocity and update rate (referring to the model accuracy as explained above) and a maximum radius for the end point search. Application specific are also criteria for evaluating the plausibility of a detected point on the structure. This is typically a local color distribution, gradient or other descriptor

that can be evaluated with respect to an optimum. The result is a trust value for a node to be part of the desired structure.

The first step in the algorithm is specifying a starting point. This is a point on the structure (with orientation) plus its local descriptor, e.g., a histogram of its neighborhood region. Either a global search in the perception field gives this starting point for the procedure or the point is given from an external process.

A radial (or spherical) search at the maximum radius gives the first guess(es) for an end point. The generated path from the starting point to the end point is accepted for a plausibility

$$\forall i \in n : P_i > P_{min}, \tag{5.1}$$

where n is the number of intermediate (way) points and P_{min} is the minimum plausibility for acceptance. If the condition is satisfied, the path (its nodes) are accepted for the online model and the end point becomes a new starting point for the next iteration. If not, other guesses on the current search radius are evaluated first, if there are none left, the search radius is decreased to its half and the procedure starts again. The algorithm terminates, if the radius would be less than the internode distance or the new end point has already been a starting point before.

The advantage of the algorithm is clear. The system does not need to do a radial/-spherical search at the internode distance in every step, but it only tries to find one end point and evaluates the way points directly from the path generation (and exits the evaluation whenever the first non-accepted way point is detected). While this benefit is already significant in the 2D case, in the 3D case it becomes essential for efficient computations.

Infinitesimal Chain Object Motion and Tracking

This section covers tracking of limp objects or general linear deformable structures. In particular, it introduces compositional modeling of chain motion and shows how this can be integrated into an established optimization approach.

"Walking" along a chain in a general mathematical sense involves alternating operations of translation (moving along a link) and rotation (changing orientation according to the node orientation)

$$T = R_n T_n \cdot R_{n-1} T_{n-1} \cdot \ldots \cdot R_0 T_0, \tag{5.2}$$

where T corresponds to the transformation or pose (position and orientation) of the n-th node relative to an arbitrary home pose of node 0. Note that for constant

velocity profiles $\forall i, j \in n : T_i = T_j$ is valid in the local reference frame of each node i and j, and refers to a translation along the local x-axis (and the value refers to the constant internode distance). These step transformations can be combined pairwise into a single transformation $T_n = R_n \cdot T$, and are initially known for each node from the recognition algorithm. The T_n correspond to the DH parameters in robot mechanics as discussed in Section 2.3.1.

In general in a tracking system a node $i \in n$ is allowed to move freely[15] in a close range to its current pose. Mathematically this refers to a transformation on the node pose close to the identity. In Section 2.3.2, methods were described to model such transformations efficiently in terms of group theory and Lie algebras. Consequently, it makes sense to apply this methodology to the tracking problem investigated here.

The basic idea with this respect is to recycle the modeling of a serial manipulator motion with the *Product-of-Exponentials*, and use it to describe the motion of a deformable linear structure in the perception domain.

Consider the pose of a node n, then the screw motion $T(\boldsymbol{\theta})$ of the node, in analogy to the PoE Equation (2.57), with orientation changes θ_i of previous nodes, can be written immediately for the 2D case as

$$T(\boldsymbol{\theta}) = \prod_{i=0}^{n} T_i(\theta_i) = \prod_{i=0}^{n} \exp(\theta_i G_i), \tag{5.3}$$

where G_i are the (motion) screws of previous nodes in the chain model. Or, referring to Equation (2.58), the pose with respect to an explicit previous pose $T(\boldsymbol{\theta})$[16].

$$T = \left[\prod_{i=0}^{n} \exp(\Delta\theta_i G_i) \right] \cdot T(\boldsymbol{\theta}) = T(\boldsymbol{\Delta\theta} + \boldsymbol{\theta}). \tag{5.4}$$

So, in order to apply a compositional object tracking method to this domain the motion screws G_i have to be found for each node in the chain. For the 2D case, this of course works analogously to the method described in and Section 2.3.3, because each node only has a single orientation parameter as its degree of freedom (like revolute joints in the serial manipulator case).

For the 3D case, it is possible to parametrize rotations conducted by ball-and-socket joints (which correspond to the node-model derived for $SE(3)$ above) by rotations about any two orthogonal axes in a plane perpendicular to the major axis of the node

[15] In more sophisticated systems node motion might be modeled to comply with a dedicated motion model.

[16] The vector $\boldsymbol{\theta}$ refers to the curvature, i.e., the orientation angles for all nodes $i \in n$ of the chain in the previous state.

in its zero position[17]. This is possible, because according to [Grassia 1998] it is not necessary to allow spinning of the joint around its major axis (see Figure 5.10). The orthogonal axes can be represented as unit vectors forming a 2D basis and thus a node rotation can be expressed with the last two infinitesimal generators in Equation (A.42), i.e.,

$$G_{R_y} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad \text{and} \quad G_{R_z} = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{5.5}$$

Defining the y- and z-axis as a basis for the rotation simplifies the next steps, but in principle any other orthogonal pair of vectors in the plane could be used just the same. The two parameters for a node are then the scales of these generators in the exponential map. If the all node pose changes need to be expressed in the reference frame of a single node (most likely the first one in the chain), for example for a tracking algorithm, the motion screws corresponding to these generators have to be derived. This is almost trivial, as it is only requires minimal modification of the procedure in Section 2.3.3.

$$G_{\boldsymbol{s}} = \begin{pmatrix} [\boldsymbol{x}]_\times & \boldsymbol{y}^T \\ 0 & 0 \end{pmatrix}, \quad \text{i.e.,} \quad \boldsymbol{s} = (\boldsymbol{x}, \boldsymbol{y}) \tag{5.6}$$

can be found for the y-rotation with T denoting the pose of the node

$$\boldsymbol{x} = \boldsymbol{x_p} - \boldsymbol{x_o} = T(0,1,0,1)^T - T(0,0,0,1)^T \quad \text{and} \quad \boldsymbol{y} = \boldsymbol{x_o} \times \boldsymbol{x} \tag{5.7}$$

and for the z-rotation, as already derived in Section 2.3.2,

$$\boldsymbol{x} = \boldsymbol{x_p} - \boldsymbol{x_o} = T(0,0,1,1)^T - T(0,0,0,1)^T \quad \text{and} \quad \boldsymbol{y} = \boldsymbol{x_o} \times \boldsymbol{x}. \tag{5.8}$$

The pose of some node after one cycle in the algorithm thus is in the 3D case, as an extension to Equation (5.4) presented earlier, can be written as

$$T = \left[\prod_{i=0}^{n} \exp(\Delta\theta_i^y G_i^y) \exp(\Delta\theta_i^z G_i^z) \right] \cdot T(\boldsymbol{\theta}) = T(\boldsymbol{\Delta\theta} + \boldsymbol{\theta}) \tag{5.9}$$

This is an elegant method for modeling possible motion of a linear deformable structure with as little as one parameter per node in $SE(2)$ or two parameters per node

[17] Here, the major axis is chosen to be the x-axis so that the following (link-)translation corresponds to a pure displacement on the x-axis in analogy to the 2D case.

in $SE(3)$. In addition, these parameters have small values for orientation changes, because the identity transformation, i.e. $\theta_i = 0$, refers to no change of the node orientation (or curve element in the chain model).

Still tracking the object is not a trivial task. At least, the algorithm has to optimize, e.g., perform predictions and corrections in a Bayesian tracking process, on n in parameters $SE(2)$ or $2n$ in parameters $SE(3)$. Luckily the values sought by the tracking system only range in a short interval around zero for each parameter - which is due to the infinitesimal motion model. This greatly reduces rounding errors and speeds up the process.

Also the algorithm is not designed to find a global optimum for all parameters at once, but instead, beginning with a start point in the chain model, the node parameters are optimized sequentially. There is a vast amount of literature on such optimization and Bayesian tracking. The algorithm presented here uses the Bayesian tracking pipeline implemented in the *OpenTL* visual-tracking framework. Please refer to [Panin et al. 2008] for a detailed introduction.

Preliminary Application: SFB Workpiece Analysis

Workpiece analysis units perform several visual tasks on the camera images provided from low-level units. A new workpiece detection unit continuously observes the workspace using the top-view camera and applies the early information processing and attention condensation strategies from Section 4.2. The unit fires an event if a new workpiece is present on the table.

A structure analysis unit uses the algorithms from Section 5.4.1 to build a model of the object online and determine possible grasping points. Also knots are detected at intersections of the linear structure (see Figure 5.12) and a strategy for resolution can be triggered by firing a "knot detected" event.

Finally, before joining the ends of the workpiece, the profile of the rubber band has to be analyzed by a special unit. The unit performs template matching on the ends to determine whether the orientation of the ends has to be adjusted before joining (see Figure 5.13).

5.4.2 Auto-Calibration

In this work, it is always assumed, that the sensorimotor systems are calibrated. This means, the transformations between sensor and actuator (in particular end-effector) reference frames are known from the setup specification. However, it would be interesting if finding such calibration information autonomously is feasible using the

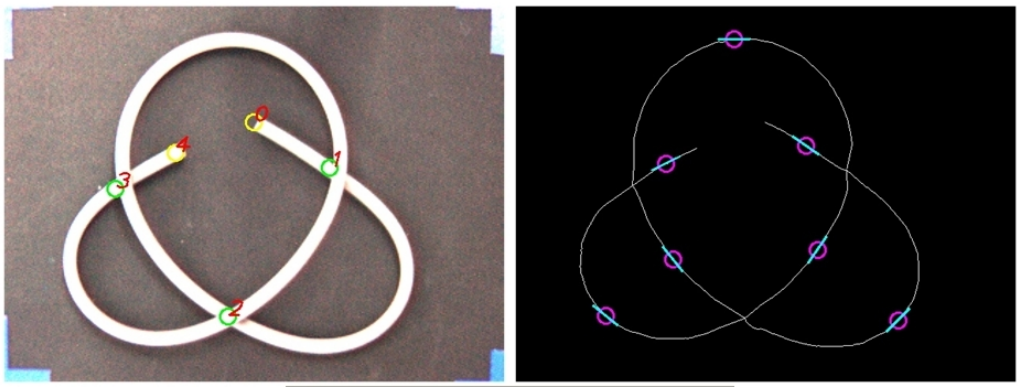

Figure 5.12 *Knot and grasping point detection result on linear structures.*

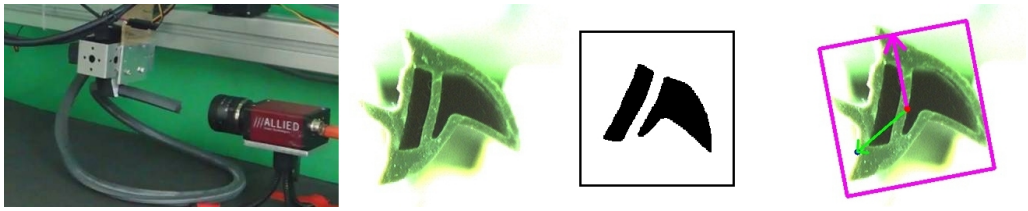

Figure 5.13 *Template matching for determining the workpiece orientation before joining the ends.*

activation features based on Lie algebras discussed in Section ?? and Section 4.2.

The key idea of the algorithm for autonomous hand-eye calibration is finding the sensor-actuator space conversion matrix (or equivalently its inverse), i.e., the spatial uplink, from features corresponding in the temporal and excitative domain. The problem then reduces to approximating a solution to an over-determined matrix equation system.

In a common setup, one could take the actuator system as a space of reference, so for any feature $F^A = F^C$ applies, as described in Section ??. The conversion matrix T then applies to

$$\boldsymbol{p}^A = \boldsymbol{p}^C = T\boldsymbol{p}^{S_j} \quad \text{or} \quad \boldsymbol{p}^{S_j} = T^{-1}\boldsymbol{p}^C = T^{-1}\boldsymbol{p}^A \tag{5.10}$$

for any activation feature $F = \{\boldsymbol{s}, \boldsymbol{p}, \boldsymbol{t}\}$. In the autonomous calibration process, the algorithm assumes a static environment or constrained environment. This means the only activation propagated from the sensor spaces originates from the part of the actuator being calibrated. This can be achieved for example by tracking a marker

attached to the end-effector[18] or by adjusting the early information filter accordingly (see SECTION 4.2.2).

For calibration it is expected that the perceptive field covers the actuator space, at least the patch where the part to be calibrated is located. In the following this part is considered to be the end-effector of a robot. Also, the process is only introduced on a single sensor, as for multiple sensors the algorithm would work analogously.

In the calibration loop, temporal consistency is essential, as any motion of the end-effector may cause direct activation in the perceptive fields of the sensory system. The calibration utilizes a property of expressing motion in terms of Lie algebra

$$T_{p\to p'} = \prod_{i=1}^{n} \exp(s_i^A G_i^A), \tag{5.11}$$

projection of scaled (by s_i^A) infinitesimal generators G_i^A into the actuator space A which is usually $SE(3)$. Within the end-effector domain each of the infinitesimal generators G_i^A is sequentially actuated, so at any time

$$\boldsymbol{s}^A = (0, \ldots, s_i^A, \ldots, 0)^T. \tag{5.12}$$

In this way, the perception (apart from noise) necessarily correlates to this actuator action directly. The scale or activation $\boldsymbol{s}^A$, i.e., the value of an activation s_i^A on a generator G_i^A is not relevant in this case as the calibration would need to normalize on activations, so the influence of an activation s_i^A distributes on the normalized activation received from the sensor $\boldsymbol{s}^S$

$$\boldsymbol{s}^A \mapsto \boldsymbol{s}^S \quad \text{with} \quad |\boldsymbol{s}^A| = |\boldsymbol{s}^S|, \tag{5.13}$$

according to some weight vector $\boldsymbol{w}^S = (w_0^S, \ldots, w_n^S)^T$,

$$\boldsymbol{s}^S = (s_0^S, \ldots, s_n^S)^T = (w_0^S s_i^A, \ldots, w_n^S s_i^A)^T. \tag{5.14}$$

Within the calibration, a *Kalman*-filter [KALMAN 1960] could be used to optimize on the weights with respect to noisy perception. Once the weights are found for translation generators, it would be also possible to determine the relative orientation of a sensor towards the actuator system in the same way. By composition of multiple sensor devices simultaneously evaluating these weights then reconstruction of the conversion matrix from sensor to actuator space (and back) would be possible.

[18] For the vision domain the libraries *OpenCV* (`http://opencv.org`) and *OpenTL* (`http://opentl.org`) provide sufficient functionality for such purposes.

5.4.3 Dual Quaternions

It would be interesting to evaluate a competitive representation of $SE(3)$, opposed to decoupled or matrix representations utilized in this work, the dual quaternion representation. Only recently, dual quaternions have been (re-)introduced for describing transformations, rigid body motions, interpolations, and the like. Dual quaternions are a composition of dual numbers [Clifford 1873, Study 1891] and quaternions. While quaternions (see Appendix A) are suitable to represent rotations in 3D space by means of the four dimensional unit hypersphere $S(3)$, dual quaternions represent both rotations *and* translations in 3D space by means of a four dimensional hyper-complex sphere.

The basis of this extension is the *Principle of Transference* [Selig 2005]. This essentially states, that any representation of rotations in 3D space $SO(3)$ can be transferred into a representation of rotations and translations in 3D space $SE(3)$ by means of using dual numbers instead of real number within the representation.

Dual numbers extend real numbers with an additional imaginary entity ε with the special property $\varepsilon^2 = 0$ (in contrast to the "normal" imaginary number i with $i^2 = -1$). The element ε thus is *nilpotent* and dual numbers are defined by

$$\check{z} = a + b\varepsilon \quad \text{with} \quad a, b \in \mathbb{R} \tag{5.15}$$

A dual quaternion for example then, instead of being defined as $\boldsymbol{q} = w + x\mathrm{i} + y\mathrm{j} + z\mathrm{k}$ with imaginary units i,j,k, is defined as

$$\check{\boldsymbol{q}} = \check{w} + \check{x}\mathrm{i} + \check{y}\mathrm{j} + \check{z}\mathrm{k}, \tag{5.16}$$

where $\check{w}, \check{x}, \check{y}$ and $\check{z}$ are dual numbers instead of real values as in $\boldsymbol{q}$. Without further discussion, it is stated, that this yields interesting geometric properties and might further simplify algorithms in the action, perception and coordination domain of sensorimotor systems.

Appendix A

Group Representations for Robotics

Contents

In Chapter 2 some background on group theory in general was given. The basic concepts of Lie groups, Lie algebras, infinitesimal generators and exponential mappings were explained alongside. However, the introduction only covered representing transformation groups by means of transformation matrices. Yet, there may be more efficient representations regarding applications to robotics and sensorimotor systems. Such efficient representations are thus derived in the following.

For robot applications, we basically consider a space of three dimensions, as we (and all the robots we are able to build) live there. In 3D space in principle three types of transformations are possible on (rigid) objects: translations, rotations, and reflections. Concerning a real-world machine such as a robot, we can discard reflections, as they can not be physically performed [Selig 2003]. Thus the elements of the transformation group for robotics are composed from 3D rotations and translations. It is common to use an abbreviation for this type of group, $SE(3)$, which stands for ***S**pecial **E**uclidian* group of three dimensions. The group is called "Euclidian", because the transformations preserve angles and proportions of the objects being transformed (as opposed to e.g., a perspective projection). It is called "Special", because we refrain from allowing reflections. One common notation for transformations of $SE(3)$ utilizes 4×4 matrices

$$T_{SE(3)} = \begin{pmatrix} R & \boldsymbol{t} \\ 0 & 1 \end{pmatrix}, \tag{A.1}$$

where R denotes a 3×3 rotation matrix and $\boldsymbol{t}$ a translation vector. Applying such transformations (or short "transforms") to points of our 3D coordinate world, we can find a consistent description of any pose a rigid object can possibly take. The term *pose* or *posture* stands for a specific orientation and position of an object in space.

Nevertheless, group theory knows many classifications of transformation groups in arbitrary dimensions, and in fact, this section will introduce the ones relevant not only for robotics, but also for computer vision and perception (see TABLE A.1).

		Translations	**Rotations**	**Combined**
2D	Lie Group	$\mathbb{R}^2$	$SO(2)$	$SE(2)$
	Lie Algebra	trivial	$so(2)$	$se(2)$
	Group-Dimension	2	1	3
	Representation (e.g.)	2D vector	scalar angle	3D vector
3D	Lie Group	$\mathbb{R}^3$	$SO(3)$	$SE(3)$
	Lie Algebra	trivial	$so(3)$	$se(3)$
	Group-Dimension	3	3	6
	Representation (e.g.)	3D vector	3×3 matrix	4×4 matrix

Table A.1 *Overview of relevant groups for robotics and computer vision. The group dimension corresponds to the number of infinitesimal generators of the algebra.*

Considering the field of visual processing, in particular regarding detection, recognition and tracking of objects present in the field of view, *sometimes* the set of relevant transformations can be restricted to two dimensions. This is the case, because processing actually takes place in a 2D image plane into which elements of the physical world are projected[1]. Thus, "sometimes" here refers to the algorithm applied for visual processing. Typically these algorithms operate either in image space (2D) or object space (3D). However, hybrid approaches are possible, e.g., hierarchical tracking systems, where low-level features are tracked in image space and higher level object representations are tracked in 3D space.

The problem of representation corresponds to choosing a proper suitable general (mathematical) description of a set of transformations relevant for a specific task. As mentioned above, for robot control applications only transformations corresponding to rotations and / or translations in 3D are of interest, while perceptive tasks like visual processing may be performed in 3D or 2D representations.

In general, there is no "best choice" for this description. Possibilities for 3D space (which typically requires six degrees of freedom) include decoupled descriptions such as a translation vector and a quaternion for rotations, or combined representations

[1] This also holds for other sensory devices, where the projection might take place in a different dimensional sensor space.

like the most wide-spread 4×4 matrices, screw motions, or even dual quaternions (e.g. [Kavan et al. 2006]). For each of those advantages and disadvantages apply regarding redundancy, computational complexity, accuracy and numerical stability; or from a more technical viewpoint, means for combination, interpolation, or scaling, to name but a few.

In the sections below, only representations applying to 3D space are discussed in greater detail[2], because simplifying the representations to 2D space is straightforward or even trivial. With respect to their practical relevance, three different representations of transformations in $SE(3)$ are explained, namely decoupled representations T_D, transformation matrices T_M, and screw motions T_S. Where necessary, also the construction methodology for the Lie algebra of the presented transformation representation is described.

A.1 Decoupled Representations

In terms of group theory, $SE(3)$ comprises the semidirect product of the group of translations $\mathbb{R}^3$ and the group of rotations without reflections $SO(3)$[3], the ***S**pecial **O**rthogonal* group of three dimensions,

$$SE(3) = \mathbb{R}^3 \ltimes SO(3). \tag{A.2}$$

This means, naturally, rotations denoted by $SO(3)$ and translations $\mathbb{R}^3$ act on the same space, the 3D physical world, not disjunct spaces[4]. The decoupled representations of $SE(3)$ exploit this feature of the structure and keep the both parts of a transformation separate. Because this is intuitively clear to most people, decoupled representations of transformations are utilized frequently. The representation with separate rotation $\boldsymbol{r}$ and translation $\boldsymbol{t}$ parts of a transformation, can be denoted as

$$T_D = \langle \boldsymbol{r}, \boldsymbol{t} \rangle. \tag{A.3}$$

For many practical applications this also allows for significantly simplified computations. For example, consider a linear trajectory of a robot end effector. Computing any intermediate transformation would then only require a simple scaling on the

[2] Also, whenever the term *transformation* is used, it refers to a transformation in $SE(3)$, i.e., a rotation, a translation or a combination of the two in 3D space.

[3] The general group of 3D rotations also considering reflections is called orthogonal group $O(3)$.

[4] To be exact, the semidirect product specifies, that only the $SO(3)$ acting on $\mathbb{R}^3$ makes sense, while the other way round would not. With a direct product instead, swapping the parts of the product would not yield an undefined, but an isomorphic structure.

delta translation vector $\delta\boldsymbol{t} = \boldsymbol{t}_{end} - \boldsymbol{t}_{start}$ of the transformation, while the rotation part could remain constant.

$$\forall i \in n : T_i = \langle \boldsymbol{r}, \frac{i}{n}\delta\boldsymbol{t} + \boldsymbol{t}_{start} \rangle, \tag{A.4}$$

where n is the number of intermediate transformations (way points).

Considering the rotation part, one can in particular choose an adequate representation according to the desired task, be it axis-angle description, quaternions, rotation matrices, to name but the most common. Also it is easily possible to switch between any of those representations using common efficient algorithms.

The challenge with decoupled representations appears in the synchronization of both parts of the transformation tuple, e.g. for blending or interpolation between transformations applying rotation and translation simultaneously. Consider for example a constant robot end-effector linear and angular target velocity. Say we are moving along a trajectory and both rotation and translation should start and finish at the same time yielding a smooth motion. In practice, it is very unlikely that both, rotation and translation, require the same amount of interpolations to meet the desired target velocity. The developer of the system thus first has to compute the intermediate number of steps for rotations and translations and then choose the maximum of both numbers and compute the next pose on the trajectory accordingly (see more on smooth online trajectory generation in Section 4.1).

Also, from the point of view of a software engineer, the decoupled representation requires some additional effort, as both parts have to be stored, monitored, and maintained synchronously. In practice, when decoupled representations are used, mostly the separate parts are described using a unique representation internally (e.g., quaternions for rotation, 3D vector for translation), and means for access of the various rotation representations are provided through the API. However, frequent conversions between internal and external representation eventually yields inefficient code, and the advantage of decoupling may vanish completely.

The next paragraphs investigate several practically relevant representations of both parts of a decoupled representation of $SE(3)$. For translations, this is the 3D (homogeneous) vector representation and the matrix representation. For rotations $\in SO(3)$, rotation matrices, axis-angle representations, and quaternions are discussed. Any of those can be combined to specify a transformation in $SE(3)$ with $T_D = \langle \boldsymbol{r}, \boldsymbol{t} \rangle$ defined above.

A.1.1 Translation

Usually the translation part of a 3D transformation is represented as a vector $\boldsymbol{t} \in \mathbb{R}^3$

$$\boldsymbol{t} = (t_x, t_y, t_z)^T. \tag{A.5}$$

Applying the transformation to a point $\boldsymbol{p} = (p_x, p_y, p_z)^T$ with three coordinates in space then only corresponds to a simple vector addition

$$\boldsymbol{p}_1 = \boldsymbol{t} + \boldsymbol{p}_0. \tag{A.6}$$

However, it is also common to describe points in *homogeneous* coordinates. In this case, an additional dimension is constructed which only contains a single scalar 1. Then, applying translations to a point can also be expressed by means of a matrix multiplication,

$$\boldsymbol{p}' = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} \boldsymbol{t} + \boldsymbol{p} \\ 1 \end{pmatrix} = \begin{pmatrix} p_x + t_x \\ p_y + t_y \\ p_z + t_z \\ 1 \end{pmatrix}. \tag{A.7}$$

It is obvious, that matrix and vector representations of translations are equivalent (apart from the "artificial" additional dimension).

The basis vectors of the translation vector space are the unit vectors along the coordinate axes

$$\boldsymbol{t}_x = (1,0,0,1)^T, \quad \boldsymbol{t}_y = (0,1,0,1)^T, and \quad \boldsymbol{t}_z = (0,0,1,1)^T. \tag{A.8}$$

The elements of the Lie group for translations in 3D space are composed by addition of scaled versions of those vectors.

$$\boldsymbol{t} = x\boldsymbol{t}_x + y\boldsymbol{t}_y + z\boldsymbol{t}_z \tag{A.9}$$

The elements of the Lie algebra for pure translations are composed from the infinitesimal basis generators as usual. These generators can be obtained from the (partial) derivative of the basis vectors of $\boldsymbol{t}$ at the identity

$$G_x = \frac{\partial \boldsymbol{t}}{\partial t_x}\Big|_{t_x=0} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad G_y = \frac{\partial \boldsymbol{t}}{\partial t_y}\Big|_{t_y=0} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad G_z = \frac{\partial \boldsymbol{t}}{\partial t_z}\Big|_{t_z=0} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}. \tag{A.10}$$

Elements of the Lie algebra are hence scaled versions of these infinitesimal generators composed by addition,

$$\boldsymbol{t_{Lie}} = xG_x + yG_y + zG_z, \tag{A.11}$$

and the exponential map transfers these back into the Lie group of translations in $\mathbb{R}^3$,

$$\boldsymbol{t} = \exp(\boldsymbol{t_{Lie}}) = \exp(xG_x + yG_y + zG_z). \tag{A.12}$$

A.1.2 Rotation by Matrix

When it comes to rotations, i.e., changes of the orientation of a rigid body in 3D space, things get a little more complicated. Rotations in 3D space are generally part of the group $SO(3)$, the special orthogonal group. The most common representation of rotations uses 3×3 matrices. The rotation R is specified by a set of three orthonormal unit basis vectors $\boldsymbol{b} \in \mathbb{R}^3$ of the new, rotated system,

$$R_{3\times3} = (\boldsymbol{b_0}^T, \boldsymbol{b_1}^T, \boldsymbol{b_2}^T), \quad \text{with} \quad \boldsymbol{b_0 b_1} = 0 \text{ and } \boldsymbol{b_0} \times \boldsymbol{b_1} = \boldsymbol{b_2} \tag{A.13}$$

Considering this representation, almost arbitrary rotations can be composed of a sequence of rotations around the "old" coordinate frame axes (composition of *Euler* angles)

$$\begin{aligned} R_x(\theta) &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}, \\ R_y(\theta) &= \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}, \\ R_z(\theta) &= \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}. \end{aligned} \tag{A.14}$$

However, several issues arise when this method for creating the representation is used: first, the composition is non-commutative. It does matter which rotation is applied first. Then, so called *gimbal-locks*[5] may occur. The reason is the alignment of gimbal axes in certain configurations and the resulting loss of a degree of freedom. Nevertheless, the main advantage of the matrix representation of rotations is that they can be combined easily by matrix multiplication $R = R_n \cdots R_1 R_0$.

[5] Wikipedia gives a good introduction to the problem of gimbal-locks at `http://en.wikipedia.org/wiki/Gimbal_lock`

Concerning the elements of the Lie algebra for $SO(3)$, usually denoted by lower case letters $so(3)$, the approach is analogous to the group of translations. Imagine the three rotations around the coordinate axes, one after the other (*Euler* composition) as specified above. The derivatives, evaluated at the identity $\theta = 0$ then are

$$G_{R_x}(\theta) = \frac{dR_x}{d\theta}\bigg|_{\theta=0} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -\sin\theta & -\cos\theta \\ 0 & \cos\theta & -\sin\theta \end{pmatrix}\bigg|_{\theta=0} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}, \tag{A.15}$$

$$G_{R_y}(\theta) = \frac{dR_y}{d\theta}\bigg|_{\theta=0} = \begin{pmatrix} -\sin\theta & 0 & \cos\theta \\ 0 & 0 & 0 \\ -\cos\theta & 0 & -\sin\theta \end{pmatrix}\bigg|_{\theta=0} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{pmatrix}, \tag{A.16}$$

and

$$G_{R_z}(\theta) = \frac{dR_z}{d\theta}\bigg|_{\theta=0} = \begin{pmatrix} -\sin\theta & -\cos\theta & 0 \\ \cos\theta & -\sin\theta & 0 \\ 0 & 0 & 0 \end{pmatrix}\bigg|_{\theta=0} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}. \tag{A.17}$$

In general

- elements of $SO(3)$ specifying an arbitrary rotation around one of the axes (according to the orthonormal basis of the rotation space) can be obtained with a scale $s \in \mathbb{R}$, as
$$T_R = \exp(s \cdot G_R),$$
- and the group elements specifying rotations around one or more axes can be obtained with the exponential of a sum of scaled generators
$$T_R = \exp\left(\sum_{i\in\{x,y,z\}} s_i \cdot G_{R_i}\right) \quad \text{with} \quad s_i \in \mathbb{R}.$$

A.1.3 Rotation by Axis-Angle

A different representation of rotations is based on *Euler's rotation theorem* [Euler 1776][6]. The theorem says, that any rotation of a rigid body with one point of the body on the origin in 3D space can be described by an axis through the origin and an angle of rotation around that axis. Basically two different notations of this are commonly in use:

[6] The Latin original reads as "Quomodocunque sphaera circa centrum suum conuertatur, semper assignari potest diameter, cuius directio in situ translato conueniat cum situ initiali".

1. A unit length axis vector $\hat{\boldsymbol{r}}$ plus an angle α specifying the rotation about the axis
$$T_R = \langle \hat{\boldsymbol{r}}, \alpha \rangle \quad \text{with} \quad |\hat{\boldsymbol{r}}| = 1.$$

2. A non-unit axis vector $\boldsymbol{r}$ with the norm of the vector representing the rotation angle α
$$T_R = \boldsymbol{r} \quad \text{with} \quad |\boldsymbol{r}| = \alpha.$$

Concerning the elements of the Lie algebra to this representation of the group $SO(3)$, the infinitesimal generators still correspond to the partial derivatives of the axis vector components as described before, which account for generating the axis of rotation. Scaling the generators then corresponds to creating the amount of rotation around the axis desired. This obviously is straight forward in the second (non-unit axis) representation from above. The first (unit-length axis) representation can be obtained using the length of the vector

$$\alpha = \sqrt{s_x^2 + s_y^2 + s_z^2}, \tag{A.18}$$

for the amount of rotation and applying an additional normalization step

$$s'_x = \alpha^{-1} s_x, \quad s'_y = \alpha^{-1} s_y, \quad \text{and} \quad s'_z = \alpha^{-1} s_z. \tag{A.19}$$

The axis-angle representation does not suffer from gimbal lock, and in its second, more concise notation avoids redundancy, as it only uses three parameters for the three degrees of freedom. Nevertheless, from a technical perspective, in particular when it comes to combining two arbitrary rotations, the axis-angle representation of rotations requires conversion to another representation. Thus, a slightly more complicated structure, namely the *quaternion*, is in general more efficient for practical applications.

A.1.4 Rotation by Quaternion

Quaternions were introduced by William R. Hamilton in 1853 [Hamilton 1853, Blaschke 1960]. They represent rotations in 3D space on the unit 3-sphere, a hyper sphere in four dimensional space denoted by S^3 which is isomorphic to $SU(2)$[7]. Of course, this is not as intuitive as the previous representations because it uses a four dimensional structure, but there are some significant advantages.

[7] $SU(2)$ is the ***S**pecial **U**nitary* group of two dimensions. Considering the correspondence to $SO(3)$, S^3 thus is a double cover: two unit quaternions actually correspond to every rotation.

The structure of a quaternion comprises four elements, a real value w (scalar part) and three imaginary values $\boldsymbol{v} = (x, y, z)$ called the vector part. The structure is then defined as

$$\boldsymbol{q} = w + x\mathrm{i} + y\mathrm{j} + z\mathrm{k}, \tag{A.20}$$

where the imaginary entities i,j,k have special properties. These properties have been first scratched into *Brougham Bridge* in Dublin by William Hamilton in 1843, as he walked by and had the ingenious idea of quaternion multiplication

$$\mathrm{i}^2 = \mathrm{j}^2 = \mathrm{k}^2 = \mathrm{ijk} = -1. \tag{A.21}$$

Quaternion multiplication can hence be derived as

$$\begin{aligned} \boldsymbol{q}_1 \cdot \boldsymbol{q}_2 = \quad & (w_1 w_2 - x_1 x_2 - y_1 y_2 - z_1 z_2) \\ + \quad & (w_1 x_2 + x_1 w_2 + y_1 z_2 - z_1 y_2)\ \mathrm{i} \\ + \quad & (w_1 y_2 - x_1 z_2 + y_1 w_2 + z_1 x_2)\ \mathrm{j} \\ + \quad & (w_1 z_2 + x_1 y_2 - y_1 x_2 + z_1 w_2)\ \mathrm{k} \end{aligned} \tag{A.22}$$

or using the scalar / vector notation

$$\boldsymbol{q}_1 \cdot \boldsymbol{q}_2 = \langle w_1, \boldsymbol{v}_1 \rangle \cdot \langle w_2, \boldsymbol{v}_2 \rangle = \langle w_1 w_2 - \boldsymbol{v}_1 \boldsymbol{v}_2, \boldsymbol{v}_1 \times \boldsymbol{v}_2 + w_1 \boldsymbol{v}_2 + w_2 \boldsymbol{v}_1 \rangle \tag{A.23}$$

These formulae are the basis for efficient combination of rotations with fewer operations and less rounding errors compared to matrix multiplication [Dobkin 1973, de Groote 1975, Howell and Lafon 1975]. For example for sequencing two rotations a rotation matrix multiplication requires 27 scalar multiplications, while quaternion multiplication can be optimized to use only 16 such operations at most. Other operations like constant velocity interpolation are also significantly more efficient in quaternion space [Shoemake 1985]. Furthermore, even the storage requirement is less for quaternions (nine for matrix, compared to four elements for quaternion representations)[8].

In the representation of quaternions, basis-transformations of $SO(3)$ (rotations around coordinate axes about an angle θ), correspond to

$$\begin{aligned} \boldsymbol{q}_x(\theta) &= \langle \cos(\theta/2), \sin(\theta/2)(1,0,0)^T \rangle \\ \boldsymbol{q}_y(\theta) &= \langle \cos(\theta/2), \sin(\theta/2)(0,1,0)^T \rangle \\ \boldsymbol{q}_z(\theta) &= \langle \cos(\theta/2), \sin(\theta/2)(0,0,1)^T \rangle, \end{aligned} \tag{A.24}$$

[8] Some more information on the performance of quaternion operations can be reviewed from `http://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation`.

in scalar/vector notation of a quaternion. The next equations are only given for x-axis rotations, y- and z-axis formulae can be derived analogously.

$$\begin{aligned} \boldsymbol{q}_x(\theta) &= \cos(\theta/2) + 1 \cdot \sin(\theta/2)\mathrm{i} + 0 \cdot \sin(\theta/2)\mathrm{j} + 0 \cdot \sin(\theta/2)\mathrm{k} \\ &= \cos(\theta/2) + \sin(\theta/2)\mathrm{i} \end{aligned} \tag{A.25}$$

Regarded as a function of θ, a rotation about the x-axis is then defined as

$$f(\theta) = \cos(\theta/2) + \sin(\theta/2)\mathrm{i}, \tag{A.26}$$

so the derivative $f'(\theta)$ is given by

$$\begin{aligned} f'(\theta) &= \frac{d}{d\theta} f(\theta) \\ &= \frac{d}{d\theta}(\cos(\theta/2) + \sin(\theta/2)\mathrm{i}) \\ &= -\frac{1}{2}\sin\left(\frac{\theta}{2}\right) + \frac{1}{2}\cos\left(\frac{\theta}{2}\right)\mathrm{i} \end{aligned} \tag{A.27}$$

and the infinitesimal generators for rotations parametrized by quaternions evaluate at the identity $\theta = 0$ to something as simple as

$$G_{\boldsymbol{q}_x} = f'(\theta)\big|_{\theta=0} = \frac{1}{2}\mathrm{i}, \tag{A.28}$$

and analogously for the other imaginary components

$$G_{\boldsymbol{q}_y} = \frac{1}{2}\mathrm{j}, \quad \text{and} \quad G_{\boldsymbol{q}_z} = \frac{1}{2}\mathrm{k}. \tag{A.29}$$

Finally, an element of the Lie algebra $so(3)$ can be composed by scaling and summing up

$$\boldsymbol{q}_{Lie} = s_x\frac{1}{2}\mathrm{i} + s_y\frac{1}{2}\mathrm{j} + s_z\frac{1}{2}\mathrm{k} \tag{A.30}$$

from real scalar angle values $\boldsymbol{s} = (s_x, s_y, s_z)^T$ – which is actually a quaternion with a vector part $\boldsymbol{v} = \boldsymbol{s}$ and a zero scalar part[9] $\omega = 0$. Group elements specifying arbitrary rotations in $SO(3)$ are obtained with the exponential of a sum of scaled generators as shown in the last sections, but now using the quaternion generators $G_{\boldsymbol{q}}$

$$\boldsymbol{q} = \exp(\boldsymbol{q}_{Lie}) = \exp\left(\sum_{i\in\{x,y,z\}} s_i \cdot G_{\boldsymbol{q}_i}\right) \quad \text{with} \quad s_i \in \mathbb{R}. \tag{A.31}$$

[9] As the structure of Lie algebra and Lie group elements is so similar, this also yields efficient technical implementations because the same data structure can be used.

To conclude the section, this inverse (exponential) mapping back from the Lie algebra to the Lie group of 3D rotations $SO(3)$ represented as quaternions has to be described coarsely. Considering a scalar part ω, and a vector part $\boldsymbol{v} = (x, y, z)^T$ of any quaternion, the exponential is in general derived from the power series definition of the exponential function (e.g. [Rudin 1986])

$$\exp(q) = \sum_{n=0}^{\infty} \frac{q^n}{n!} = e^{\omega} \left(\cos|\boldsymbol{v}| + \sin|\boldsymbol{v}| \cdot \frac{\boldsymbol{v}}{|\boldsymbol{v}|} \right) \tag{A.32}$$

with $|\boldsymbol{v}| = \sqrt{x^2 + y^2 + z^2}$ stating the L_2-norm of the vector part. This general formula for the exponential of a quaternion can now be used to derive the exponential mapping of Lie algebra elements. As described above, these elements $\boldsymbol{q}_{Lie}$ are just quaternions with a zero scalar value

$$\boldsymbol{q}_{Lie} = 0 + s_x \frac{1}{2}\mathrm{i} + s_y \frac{1}{2}\mathrm{j} + s_z \frac{1}{2}\mathrm{k}. \tag{A.33}$$

Here, we show the exponential map $so(3) \mapsto SO(3)$ for rotations being represented as quaternions on two crucial examples.

1. $\exp(\boldsymbol{q}^{\boldsymbol{0}}_{Lie}) = \boldsymbol{q}_{id}$, i.e., taking the exponential of the identity Lie algebra element $\boldsymbol{q}^{\boldsymbol{0}}_{Lie}$ with scales $\boldsymbol{s} = \boldsymbol{0} = (0, 0, 0)^T$ results in the identity quaternion applying no rotation $\boldsymbol{q}_{id} = 1 + 0\mathrm{i} + 0\mathrm{j} + 0\mathrm{k}$.
2. Taking the exponential of a scaled version of one of the infinitesimal generators of basis-rotation yields the unit quaternion of a such rotation.

For the first part, i.e., the scales $\boldsymbol{s} = (0, 0, 0)^T$ for the Lie algebra element (or the vector part of a zero scalar quaternion), it is clear, that

$$\begin{aligned}
\lim_{\boldsymbol{s}\to 0} [\exp(\boldsymbol{q}^{\boldsymbol{s}}_{Lie})] &= \exp(0 + 0\mathrm{i} + 0\mathrm{j} + 0\mathrm{k}) \\
&= e^0 \left(\cos(0) + \sin(0) \frac{\boldsymbol{s}}{|\boldsymbol{s}|} \right) \\
&= e^0(1 + 0) = 1 \\
&= 1 + 0\mathrm{i} + 0\mathrm{j} + 0\mathrm{k} \\
&= \boldsymbol{q}_{id}.
\end{aligned} \tag{A.34}$$

Here the limit has to be used because of the factor $\boldsymbol{s} \cdot |\boldsymbol{s}|^{-1}$. The result is true, because $\forall i \in x, y, z : |\boldsymbol{s}| \geq s_i$.

For the second part, here only given for rotations of $s_x = \theta$ around x-axis with the generator $G_{\boldsymbol{q}_x} = \frac{1}{2}\mathrm{i}$ as stated in Equation (A.28), we can write the corresponding

quaternion $\boldsymbol{q}^{\boldsymbol{s}}_{Lie}$ for the Lie algebra element $\boldsymbol{s} = (\theta, 0, 0)^T$

$$\begin{aligned} \boldsymbol{q}^{\boldsymbol{s}}_{Lie} &= \theta G_{\boldsymbol{q}_x} \\ &= \frac{\theta}{2}\mathrm{i} \\ &= 0 + \frac{\theta}{2}\mathrm{i} + 0\mathrm{j} + 0\mathrm{k} \end{aligned} \tag{A.35}$$

and may derive the Lie group element for the rotation around x-axis about an angle θ, a unit quaternion $\boldsymbol{q}_x$ with non-zero scalar value, by exponentiation. For $\boldsymbol{q}^{\boldsymbol{s}}_{Lie}$ in quaternion notation, the vector part $\boldsymbol{v} = (\frac{\theta}{2}, 0, 0)^T$ and $|\boldsymbol{v}| = \frac{\theta}{2}$ are obvious, and thus

$$\begin{aligned} \boldsymbol{q}_x(\theta) &= \exp\left(0 + \frac{\theta}{2}\mathrm{i} + 0\mathrm{j} + 0\mathrm{k}\right) \\ &= \exp(0)\left(\cos|\boldsymbol{v}| + \sin|\boldsymbol{v}| \cdot \frac{\boldsymbol{v}}{|\boldsymbol{v}|}\right) \\ &= \cos(\theta/2) + \sin(\theta/2) \cdot 1 \cdot \mathrm{i} + 0 \cdot \mathrm{j} + 0 \cdot \mathrm{k} \\ &= \cos(\theta/2) + \sin(\theta/2)\mathrm{i}. \end{aligned} \tag{A.36}$$

So in fact, applying the exponential Equation (A.36) to scaled generators in quaternion notation Equation (A.35) yields the corresponding rotation unit quaternion as expected, as

$$|\boldsymbol{q}_x(\theta)| = \sqrt{\cos(\theta/2)^2 + \sin(\theta/2)^2} = 1. \tag{A.37}$$

A.2 Transformation Matrices

Transformation matrices are the most wide-spread means for representing transformations. Many people consider them to be most intuitive. Transformations T_M are specified as 4×4 matrices in 3D space, i.e., for $SE(3)$; or 3×3 matrices in 2D space, as they operate on homogeneous coordinates

$$T_M = \begin{pmatrix} R & \boldsymbol{t} \\ 0 & 1 \end{pmatrix}. \tag{A.38}$$

For 2D space, $\boldsymbol{t}$ is a vector of two dimensions $\boldsymbol{t} = (t_x, t_y)^T$ and R is a 2×2 matrix for rotations around the z-axis. R is thus specified as

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \tag{A.39}$$

Summary A.1 – **Decoupled Transformation Representations**

Transformations of $SE(3)$ are specified by two separate parts, translation and rotation.

1. $\mathbb{R}^3$ translations are normally represented as a translation vector.
2. $SO(3)$ rotations are mostly represented as a
   - 3×3 matrix,
   - an axis of rotation and an angle, or
   - a unit quaternion.

When combining two transformations in decoupled representations, typically first rotations are applied, then translations.

The 2D transformations T_M then simply act on points represented as homogeneous vectors $\boldsymbol{p} = (x, y, 1)^T$ with

$$\boldsymbol{p}' = T_M \boldsymbol{p} = T_M \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x\cos\theta - y\sin\theta + t_x \\ x\sin\theta + y\cos\theta + t_y \\ 1 \end{pmatrix}. \tag{A.40}$$

In 3D space, transformations T_M from Equation (A.38) are defined analogously, where R is then a 3×3 rotation matrix and $\boldsymbol{t}$ a vector $\in \mathbb{R}^3$ for translations. Rules for combination of transformations by means of matrix multiplication, as well as scaling, exponentiation and formulas for matrix derivatives can be reviewed from any ordinary textbook on mathematics, e.g., [Meyberg and Vachenauer 2001].

For $SE(3)$ one can define the following six infinitesimal generators considering a common orthogonal basis of the transformation space and derivatives at the identity analogously to what has been described earlier in Section 2.2 and Section A.1

$$G_{T_x} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad G_{T_y} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad G_{T_z} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \tag{A.41}$$

for translations and for rotations

$$G_{R_x} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad G_{R_y} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad G_{R_z} = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{A.42}$$

The Lie algebra elements are then scale vectors $\boldsymbol{s} \in \mathbb{R}^6$, or in matrix notation with $i \in \{T_x, T_y, T_z, R_x, R_y, R_z\}$.

$$G_{\boldsymbol{s}} = \sum_i (s_i \cdot G_i) = \begin{pmatrix} 0 & -s_{R_z} & s_{R_y} & s_{T_x} \\ s_{R_z} & 0 & -s_{R_x} & s_{T_y} \\ -s_{R_y} & s_{R_x} & 0 & s_{T_z} \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{A.43}$$

and the elements of the Lie group (transformations in $SE(3)$) close to the identity can simply be constructed from those algebra elements with

$$T_M = \exp(G_{\boldsymbol{s}}). \tag{A.44}$$

Numerous algorithms regarding common operations, i.e., in particular exponentiation, on this mathematical structure exist and are publicly available (e.g., [Lawson 1967, Higham 2004]). Still, matrices encounter disadvantages regarding computational complexity, e.g. when derivatives, blending or interpolation operations have to be performed - if they not yield sub-optimal or inaccurate solutions at all (e.g., [Kavan et al. 2006]).

A.3 Screw Motions

Another notation of transformations using matrices uses the concept of screw motion. Screw motions intrinsically provide for interesting properties for robotics. They are most suitable for describing the action of revolute joints (e.g. in serial manipulators), as transformations can be parametrized by a single angle of rotation and computations then become efficient and mathematically elegant. Nevertheless, understanding the concept is non-trivial for most people and also general application sometimes seems cumbersome.

In essence they are based on *Chasles's theorem* also called *Mozzi's theorem* [Jackson 1942], an extension of *Euler's rotation theorem* [Euler 1776], which states an alternative way of describing a transformation. It basically says, that whatever

motion a physical rigid object performs in 3D space, one can also view this as a rotation about an axis and a translation about the same axis[10]

$$T_S = \begin{pmatrix} R(\theta, \boldsymbol{x}) & \frac{\theta p}{2\pi}\boldsymbol{x} \\ 0 & 1 \end{pmatrix}. \tag{A.45}$$

We take a closer look at the entries of the above matrix. First, the 3×3 matrix $R(\theta, \boldsymbol{x})$ specifies a rotation of θ in 3D space around the axis defined by the unit vector $\boldsymbol{x}$. The matrix is in fact derived from the common axis angle representation of a rotation. The *Rodrigues formula*[11] tells, how to obtain the rotation matrix from the axis $\boldsymbol{x}$ and angle θ

$$\begin{aligned} R(\theta, \boldsymbol{x}) &= \exp(\theta[\boldsymbol{x}]_\times) \\ &= I + [\boldsymbol{x}]_\times \sin\theta + [\boldsymbol{x}]_\times^2 (1 - \cos\theta). \end{aligned} \tag{A.46}$$

Here, I is the identity and the operator $[\cdots]_\times$ transforms a 3D vector into a 3×3 anti-symmetric matrix, such that $[\boldsymbol{x}]_\times \boldsymbol{y} = \boldsymbol{x} \times \boldsymbol{y}$, i.e.,

$$[\boldsymbol{x}]_\times = \begin{pmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{pmatrix}. \tag{A.47}$$

Analyzing $\frac{\theta p}{2\pi}\boldsymbol{x}$ from Equation (A.45), we find this a 3D vector of translation along the unit vector $\boldsymbol{x}$ with some scaling according to an additional pitch p, that expresses the actual displacement in $\boldsymbol{x}$-direction after a full rotation of 2π, and the scalar θ stretching the unit vector $\boldsymbol{x}$.

As mentioned earlier, so far only screw motions were examined, where the axis passes through the origin. Luckily though, generalizing the above results to any screw motion axis in 3D space can be easily achieved by first translating some point $\boldsymbol{y}$ on the axis to the origin, then applying the screw motion, and finally translating it back to its original position[12]

$$\begin{aligned} T_S &= \begin{pmatrix} I_3 & \boldsymbol{x_o} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} R(\theta, \boldsymbol{x}) & \frac{\theta p}{2\pi}\boldsymbol{x} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} I_3 & -\boldsymbol{x_o} \\ 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} R(\theta, \boldsymbol{x}) & \frac{\theta p}{2\pi}\boldsymbol{x} + (I_3 - R(\theta, \boldsymbol{x}))\,\boldsymbol{x_o} \\ 0 & 1 \end{pmatrix} \end{aligned} \tag{A.48}$$

[10] The formula in Equation (A.45) is of course only true if the axis $\boldsymbol{x}$ passes through the origin. A generalized version of the equation is presented in Equation (A.48).

[11] Interestingly this also defines a closed form of the exponential map from the Lie algebra $so(3)$ to the Lie group of rotations in 3D $SO(3)$.

[12] The simplified formula Equation (A.48) is adopted from [Selig 2005].

Putting it together, we generally call transformations T_S represented in this way a *screw motion*. Once the motion parameters, namely the axis of rotation $\boldsymbol{x}$, pitch p, and displacement $\boldsymbol{x_o}$ are known, the transformation can be described by a single parameter θ corresponding to the angle of rotation. This is a convenient representation for transformations conducted by a revolute joint.

Now, as screw motions are commonly written in matrix notation, also the Lie algebra and its generators, i.e., the partial derivatives of the transformation space basis evaluated at the identity, can be constructed analogously. Looking at Equation (A.46), and Equation (A.47), this is obviously similar to the rotation generators from Equation (A.42), apart from a normalization factor θ. So for example, $s_{R_x} = \theta x_1$ and so on.

The same holds for translation generators, which correspond to the displacement vector $\boldsymbol{x_o}$ and its derivative at the identity with respect to a parameter θ. In fact, this parameter is of such importance, as it is the only variable concerned for a revolute joint in robotics. This was explained in sufficient detail in Section 2.3.

Appendix B

Software Framework

Contents

THE software framework proposed in this chapter combines promising ideas of recent neuroscientific research with a blackboard information storage mechanism and an implementation of the multi-agent paradigm. Therewith, the integration framework for sensorimotor applications strives for increasing flexibility in development of robotic, computer vision, and machine intelligence applications (Section B.1). The goal is to provide a sound and easy-to-use, but yet efficient and highly parallelizable base architecture for complex sensor-based robotic systems with a strong focus on intrinsically dynamic (e.g., vision-based industrial automation) scenarios (Section B.2).

Section B.3 briefly explains the software development kit and the deployment process and finally, Section B.4 describes necessary steps to setup the software framework and shows how to build a simple, distributed application.

B.1 Architecture

Detached from physical hardware specifications, the basic building blocks (or primitive modules) may involve low-level actuator control actions just as well as sensory (for example visual) input processing or reasoning tasks. These task processing primitive blocks are called *processing unit* (PU) in the proposed framework (Section B.1.1).

Typically these semantically independent processing units need to exchange data between each other in order to map an application workflow. As means for this an event-based data management mechanism is introduced in Section B.1.2. Event triggers can be used to control execution of processing units, while generic data channels maybe used to exchange information between the units in an efficient way.

To enable implementations of concrete application workflows a mechanism for hierarchical composition of processing units is presented in Section B.1.3. This involves enveloping multiple lower-level processing units into a new, more abstract unit. In this way an implicit tree representation can be generated incrementally.

By means of a failure event in the processing units it is also possible to monitor the success of an operation. In composite units (referring to a an interconnected collection of lower-level processing units) errors in a submodule can be handled internally (for example with repetition) or escalated to the next higher enveloping unit. In this way multi-layer error handling strategy can be implemented with little effort.

B.1.1 Processing Units

Basically any algorithm, calculation, or processing task in the proposed framework has to be implemented within the main loop of a processing unit. In addition to providing this main loop, PUs also implement means for generic data exchange and event-based state transitions.

In Figure B.1 one can see the general structure of a PU. It comprises task execution trigger events, namely a `Start` and `Stop` trigger. During execution of a task the state events `Success` or `Error` maybe emitted to enable other PUs react on the processing result. It is important to mention at this point, that all processing units implement a stop event listener. This listener always has priority to the current task process, so at any time a PU can be interrupted using this methodology. Of course, this might leave the unit in an undefined state. A reset function is called automatically every time the start event is received in order to provide means for handling this issue.

A processing unit may also define dedicated data channels (black in the figure). Programmatically these data channels are templated (and possibly buffered) variables, i.e., their type is evaluated during compile time of the unit. Variables needed for, or generated within a processing task initially only exist within the processing unit for read and write access. However, data channels can be connected between processing units in the same way events can be connected. Then, both read and write access on parameter data is allowed to any connected instance.

Data channels can furthermore include an event. Then, the data channel transports a signal, whenever data is written to the channel. This is extremely useful for the different supported communication types between processing units, because it facilitates implementation of publish / subscribe paradigms, data-listeners, polling,

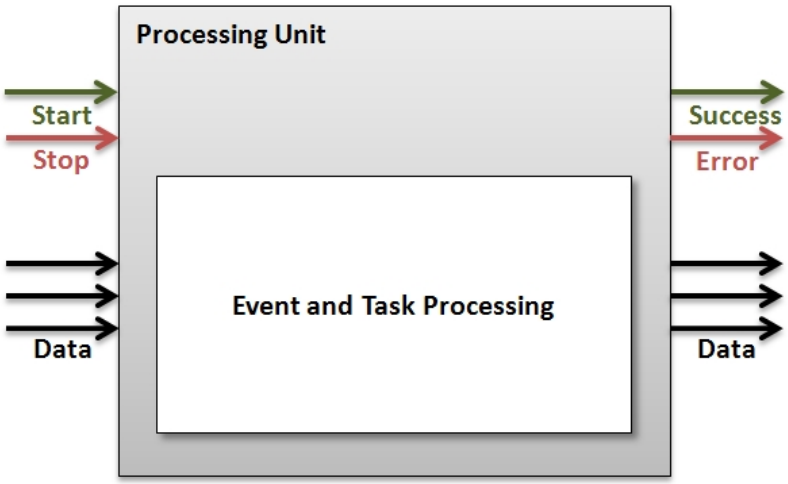

Figure B.1 *Structure of processing units.*

push services, and synchronous or asynchronous behavior.

Finally, the PU structure provides a configuration mechanism to the automation system. It is possible to load parameter defaults from a configuration file on system start-up, for example to initialize calibration parameters or define default behavior.

PUs in the framework are designed as threads. Typically, an application comprises a set of PUs, where PUs perform their action in parallel, either continuously or in a one-shot manner once they are triggered by a start event. While most hardware interfaces need a cyclic, continuous update / retrieval (such as the robot realtime interface or the camera interface), higher-level actions wrapped into a PU, e.g. moving the end-effector from A to B, handing a workpiece over to another robot, or finding a grasping point in a visual scene, most commonly only implement a non-cyclic action.

B.1.2 Data Management

In order to map a complex sensorimotor workflow, exchanging data between PUs is essential. In Figure B.2 an example (visual processing) application is shown on an abstract level. The camera device unit maps to the segmentation unit, which then maps to the visual processing unit and finally both visual processing result and camera image map to the user interface unit. In this example the camera unit thus has to share the image it acquires with the GUI and the segmentation unit, while the segmentation result has to be available in the visual processing instance. The proposed framework introduces a generic information manager for facilitating these data transfers. It is implemented as a singleton and is thus only available once in

an application instance[1].

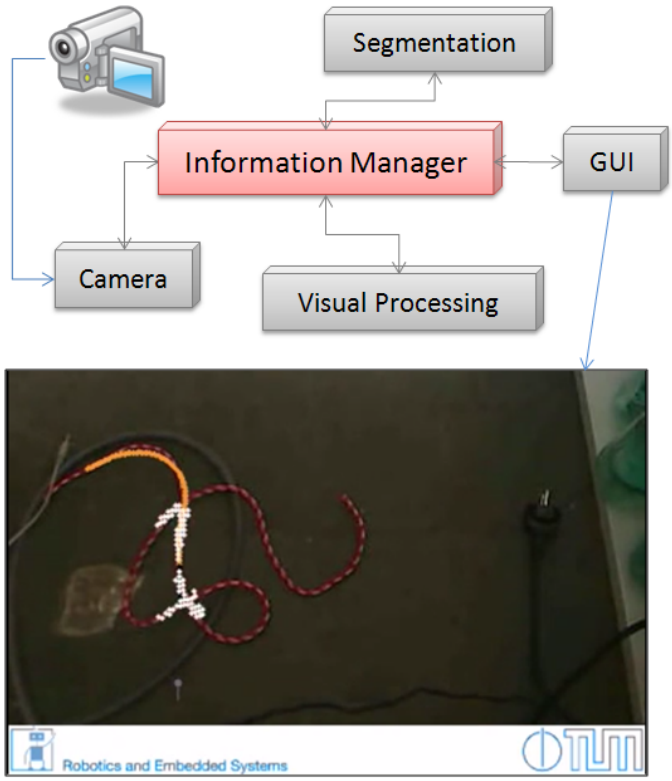

Figure B.2 *Multiple processing units (grey) coexist and communicate via the information manager (red).*

FIGURE B.3 shows the mechanism for data registration, storage, and retrieval modalities as a diagram. Before actually entering the main loop of an application (and most likely the main loop of task execution in one or more PUs) the data channels of all processing units that need to exchange data have to be connected. If for example the camera unit needs to publish the image it acquires, the corresponding data channel has to be shared. Other processing units can then connect to the channel to be able to read images from it. In general however, units connected to a data channel can always read and write data to that channel.

When connecting to a data channel, the connection can be passive or active. Active connections refer to a PU just polling data whenever it needs (second row in FIGURE B.3). If there is no data on the channel, the PU receives a `NULL` item on a

[1] In CHAPTER ?? it will be shown, that this is only true for applications run on a single computer. In a cluster setup multiple application instances and thus multiple information managers may need to synchronize across an MPI ring.

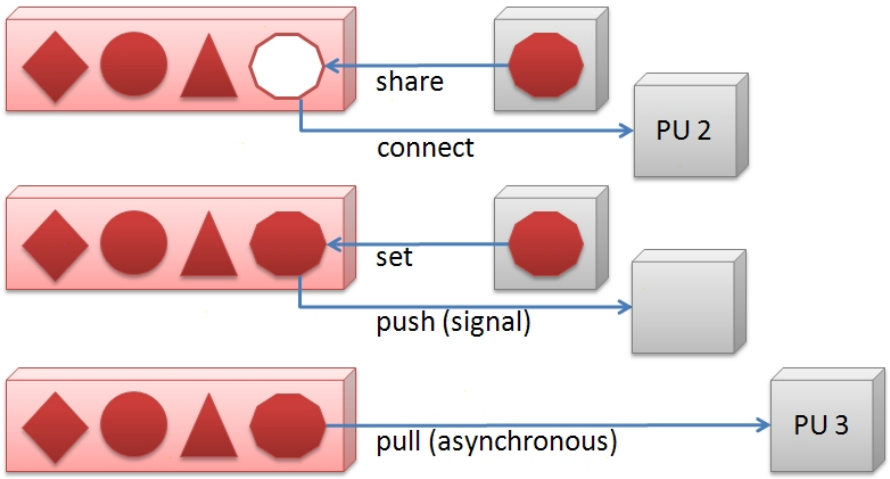

Figure B.3 *Data exchange mechanisms for processing units using the information manager.*

read operation and needs to handle this internally. This mechanism is for example useful for implementing asynchronous data access paradigms, e.g., in multi-agent systems. In a passive connection on the other hand, i.e., if a PU is connected to a channel and the connection is marked passive, whenever the data changes on the channel it emits a notification signal to the passively connected PU (third row in FIGURE B.3). Only the passively connected PU receives the signal and may then process the current data item. This mechanism is dedicated to synchronous data exchange strategies.

In the proposed implementation of the information manager a channel allows multiple active and passive connections to a data item at the same time. Thus, parts of an application implemented with the framework can be sychronous, while other parts can be asynchronous in the same program. Also, both event and data channels implement the same interface. This is useful, as it allows active connections in case there is no event associated with a data channel as described above; and, on the other hand, it allows for declaring additional input or output events to a processing unit by means of data channels. The payload of these channels then is only an event, and no additional data item.

B.1.3 Task Coupling and Abstraction

The processing units together with (a)synchronous connections between them already have all necessary means for coupling and abstraction of tasks on-board. An example is shown in see FIGURE B.4. In the figure only event channels are shown.

However, it should be clear that chaining two processing units and enveloping them into a higher-level one is easily possible also when data channels get involved. In the case depicted, no additional logic other than mapping the events from and to low-level units PU_1 and PU_2 is needed.

Of course, the task processing loop of the higher-level unit PU_{12} can be implemented to perform more sophisticated tasks than just mapping events. For example listening on computed data and producing a combined result can be implemented in the processing loop.

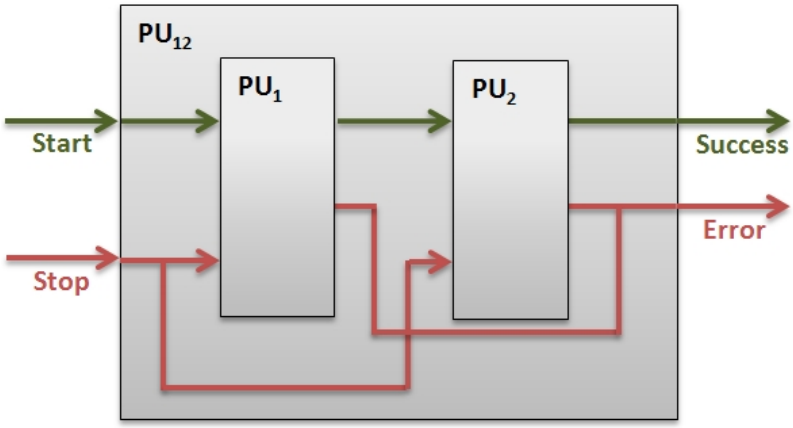

Figure B.4 *Coupling of two units into an envelope unit.*

As the composite unit PU_{12} implements the same interfaces than its components PU_1 and PU_2, it can itself again be used as a component for higher-level units. In this way units can be stacked or combined into a hierarchical structure.

The framework even implies no restriction on utilizing a unique component in more than one composites. This may lead to overlapping composite units as shown in Figure B.5. The schema in the figure for example uses the graphical user interface (GUI) unit in both the composite grasp unit and the composite analyze unit. More details on this specific example are discussed in Chapter **??**.

B.1.4 Fault Tolerance and Robustness

Considering robustness, dependability is one of the most important aspects in industrial scenarios. This is particularly important, when investigating sensor assisted robotic setups where processes are partly deterministic at best. Therefore, the aforementioned concept achieves to handle the dynamics in a sensorimotor robotic application by means of an error monitoring concept.

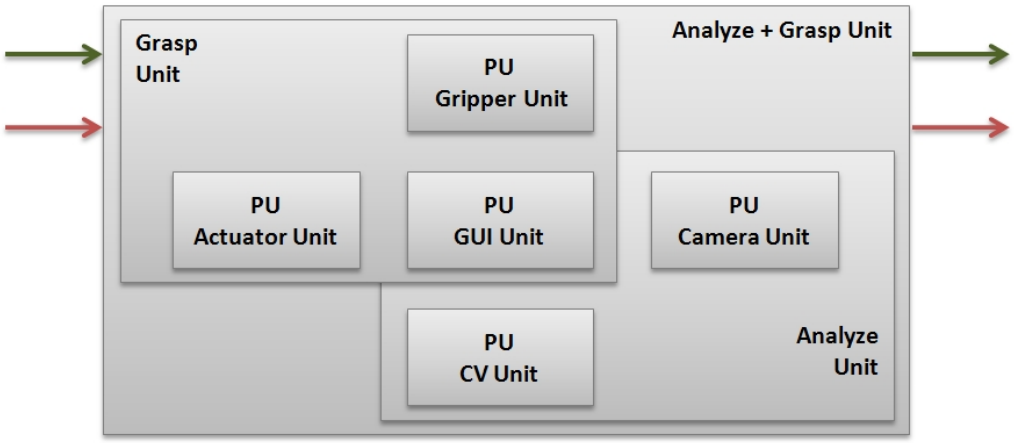

Figure B.5 *Possibly overlapping composite units can be composed from arbitrary processing units and represent abstract, higher-level task building blocks.*

The following three important questions with respect to faults and errors must be evaluated for any application complex built from processing units [ECHTLE 1998]:

1. *What can happen and where can it happen?*
   First, an error hypothesis is needed. It contains all possible sources of a fault including design errors (specification / implementation / documentation), production errors and operation errors (mechanical, electrical faults, user mistakes, logical algorithmic mistakes, etc.). Once the possible sources of faults are identified on an abstract level, the components / modules involved in the faults have to be designed in order to enable detection and recovery of possible faults.

2. *How can a fault/error/failure be detected or avoided in one of the monitoring units?*
   While avoidance means optimizing and improving a system in advance, application of sensor based fault detection, software based detection or actuator feedback based detection has to be discussed for monitoring a process during its execution.

3. *Once occurred, how can an error be handled?*
   Methods for error handling are manifold. In general, methods are error passivation / reconfiguration (evacuation / insertion / elimination), error recovery (forward / backward recovery) or error compensation (error correction / fault masking).

The fault tolerance mechanisms developed and applied here comprise two perpen-

dicular areas of measurement. On the on side, the framework described in the above sections is the basis for the modular monitoring approach, as it provides for the decomposability of the task into software modules, represented by the processing units. The framework provides means for ensuring that the system is in a good state, as the PUs provide a failure event in case of an error. According to the specification above, answering the first question is entirely due to the application developer.

Answering the second question principally it can be stated, that the software framework proposed here is only in a very limited way able to detect hardware, algorithmic or design errors directly. Only misconfiguration of processing units (for example connection channel type mismatches), or general implementation errors (like not implementing PU interface specifications) can be detected automatically. The task processing, i.e., the implementation of main loop of a processing unit, is a black box to the framework and thus has to be handled by the application developer.

However, discussing the third question, the framework provides means for passing failures inside a component to an appropriate error handling unit. This implies, that of course, in the general case of a system crash or unhandled failures inside a processing unit, the framework application will also crash.

Error Recovery

Nevertheless, this enables the developer to generate a hierarchical error handling unit by unit composition and abstraction from low-level operations (see FIGURE B.6).

For example consider a sensor-based object detection and handling task. Processing unit PU_1 in this case is the detection unit. It tries to find an object to be grasped by the robot. If this succeeds, PU_2 gets invoked. Its duty is to move the end-effector towards the object and grasp it.

The interesting error recovery occurs in PU_3. Here, if the detection fails, the sensor position can be modified. So composing the units in a way shown in the figure enables a composite unit PU_{123} to handle a detection error internally. According to [ECHTLE 1998] this corresponds to a backward recovery or soft-reset. Only if the sensor positioning or the grasping fails, the composite unit emits the failure and a higher-level unit needs to handle it.

Failure Avoidance

A common technique for avoidance of system failures is redundancy. In this case, failures of certain task processing units are anticipated and accepted. Hence multiple instances of the same processing unit are initialized from the beginning. The

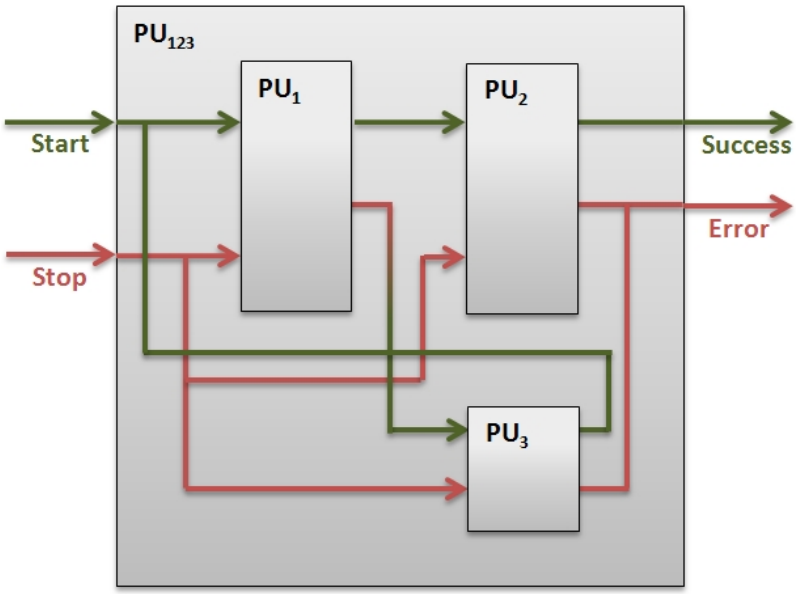

Figure B.6 *Recovering from an error within a composite unit.*

framework provides special processing units for handling the output events of such redundant instances (see FIGURE B.7).

The multiplexer units (such as PU_3) specify data channels that accept events of multiple redundant instances (PU_1 and PU_2) as an input. The multiplexer units are implemented in various variants. One for example simply suppresses errors if a `Success` event is present from one of the redundant units. Another applies a majority voting on multiple `Success` events. Other custom implementations are possible, for example to react on redundant output events and also take redundant output data channels into account. These however are application dependent and have to be implemented by the application developer.

Again, composing redundant structures into a composite unit (PU_{13}) abstracts from lower-level failure handling routines for higher-level units. Thus, hierarchical error handling can be implemented easily.

Timing Constriants

Lest we forget to mention, that also timing constraints can be considered in a multiplexing or error handling unit. This is especially useful if a task should be interrupted after some time-out interval (see FIGURE B.8).

The `Success` event of a task processing unit PU_2 is in this case monitored by a timer unit PU_1, and if no `Success` event from task processing is received in time,

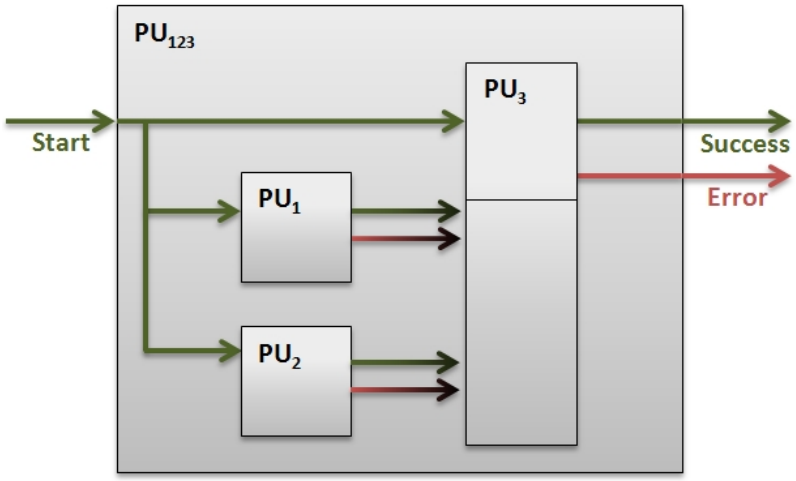

Figure B.7 *Redundancy for task completion within a composite unit.*

the timer emits a `Stop` event to the task unit and an `Error` event to higher-level handling routines. On the other hand, in case the task's `Success` event was in time, the composite unit PU_3 emits a `Success` event as well.

B.2 Parallelization and Distributed Resources

As multi-core processor architectures are unlike a few years ago already a de facto standard nowadays, state-of-the-art software frameworks *must* take parallelization techniques into account from the very beginning in the specification and design process.

In the last chapter it was briefly mentioned, that the proposed software framework is intrinsically designed for parallel application scenarios. The encapsulation of task processing into a threaded processing unit's main loop provides the means for such. Preliminary work on this topic can be reviewed from [Müller, Ziaie and Knoll 2008], while recent results are elaborated in Section B.2.1.

Additionally, the unique feature of the framework is the possibility to parallelize automatically not only on a single multi-core machine, but on a computing cluster with many multi-core members just the like (see Section B.2.2). A well established cluster computing paradigm and implementation based on the message passing interface (MPI) acts as a base communication layer and abstracts from actual cluster issues. Earlier results with this respect are discussed in [Müller and Knoll 2010, Müller et al. 2010, Müller et al. 2011].

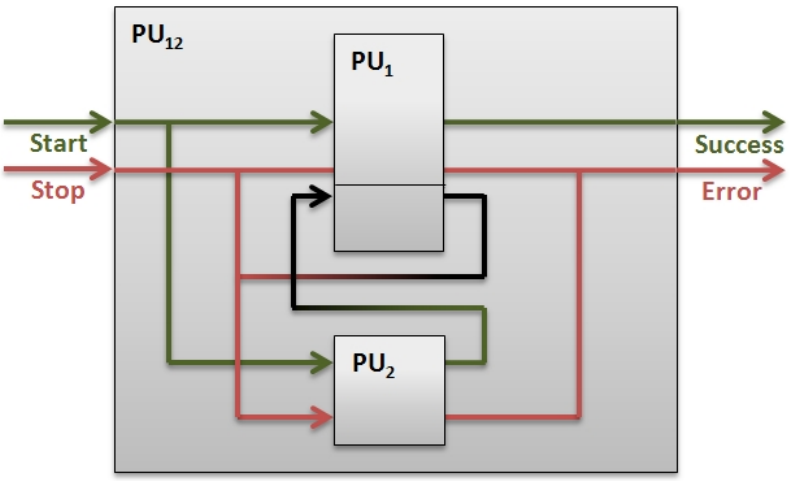

Figure B.8 *A composite unit for tasks with an additional timing constraint monitored by PU_3.*

The result is a modular system that integrates sensorbased realtime robot control with facilities for remote control. Due to the MPI base implementation of the framework also runtime distribution of tasks is achieved. Thus, for evaluation and testing the system may run on a single control PC on the one hand, and on the other hand automatically scales to multiple computers as soon as more resources become available in the MPI ring. Several student theses have sucessfully demonstrated the applicability of the proposed methods by extending the base system with multiple features integrated into new or existing processing units. For example [Hellerer 2009] modified the proposed robot control unit to use potential field based target generation and collision avoidance, [Tran 2010] worked on early visual processing and proposed a task control unit with error recovery for automation of the SFB joining task, or [Plopski 2010] developed a wireless control device using vision based recognition and active servoing.

B.2.1 Auto-Parallel Processing

Crucial to any parallel system is the choice on which hierarchical (semantic) layer of abstraction the parallelization takes place. Traditional auto-parallelization systems like *OpenMP* [Chapman et al. 2007] require the extensive usage of dedicated statements within the source code (e.g. `pragma` in OpenMP) to mark parallizable blocks like `for`-loops. The compiler interpretes the statements and the result is a parallel application. The approach works well, if one only wants to parallelize a simple sequential application without in depth knowledge of synchronization and

parallelization techniques. Other traditional approaches utilizing multithreading approaches work well, but they require expert knowledge on shared data access, function binding, avoidance of dead locks, and the like.

The system presented here encapsulates the actual parallelization from the application developer. The programmer only needs to define three essential structural components:

- *Decoupled Tasks*
  The developer decides which tasks are possible to be run in parallel, i.e., which tasks can be seen as a semantic entity. This works on different levels of semantic abstraction.
- *Shared Data*
  The data sharing mechanism via the information manager and data or event channels introduced in Section B.1.2 enables quick definition of shared data structures. The framework then automatically provides the synchronization and mutually exclusive access functionality.
- *Application Workflow*
  An application workflow is defined by its event and data flow. Essentially, this means that each task may specify, process or emit events and / or data signals. These can be used to determine the sequence of operations within a task, or subtask, or trigger operations within other task units. The concept was already introduced in Section B.1.1.

For example, considering a typical application for object recognition and tracking. The application comprises image acquisition, segmentation, recognition and tracking, and result visualization. Regarding the point of view of a parallelization engineer, these operations need not necessarily run in a sequential order, but instead can be decoupled into separate tasks. This already defines the structural components. Later, the software engineer has to implement the task logic into corresponding PUs (see Figure B.2).

Also, it has to be defined, which data structures need to be transferred from one task to another. In the example, the camera unit would need to transfer the input image to the segmentation unit, while this would need to transfer the result image to the recognition and tracking unit, and so forth. The system now allows to decouple these tasks automatically, as there is no reason not to load the next image from the camera, while the current image is still being processed. Having defined the shared data structures and event triggers, thread-safe read and write access as well as synchronization is intrinsically provided by the frameworks parallelization

capabilities. Internally, the framework realizes this functionality by implementing the information management as a "blackboard". Once a datum is shared, any `set` or `get` operation on a connected data channel automatically results in a post or pull (and required synchronization operations) on the global singleton information storage.

Finally, specifying an application workflow is straight forward, in fact it is implicitly given and does not need to be specified explicitly. In preliminary work [Müller, Ziaie and Knoll 2008] the information manager is only a global storage for data. Now, introducing event management, the storage was extended to be the instance of workflow management within an application. Any shared data item can be defined to emit a signal to any connected processing unit whenever the datum changes. By default, all processing units implement an event listener, so, reacting on incoming events enables the software engineer to conveniently introduce the desired workflow.

B.2.2 Automatic Distribution

In the last section parallel systems on a single computer using the intrinsic multi-threading capabilities of the framework were discussed. The programmer is in this case able to specify and implement parallel applications without explicitly dealing with distribution of tasks and synchronization issues. This section elaborates on the process of automatic task distribution whenever an application built with the framework runs on a computing cluster. A schematic overview of the parallelization layers for a such hardware setup is shown in Figure B.9.

The lowest layer comprises connected hardware components. These components (usually computers) need not be identical in their hardware configuration. In fact the cluster middleware used in the following is available for almost any current hardware platform and operating system. Only the computers need to be interconnected somehow (see Section B.2.3). The second layer abstracts from the actual hardware by means of processors. The processors do not necessarily correspond to the number of CPUs on a hardware, but it makes sense in a productive environment (see Section B.2.4). The highest layer is composed from processing units introduced in the previous chapter. In a distributed application each processing unit is first assigned to a processor, which in turn is assigned to a computer (i.e., the hardware infrastructure). The proposed framework handles this hierarchical assignment autonomously, except for the special case discussed in Section B.2.5.

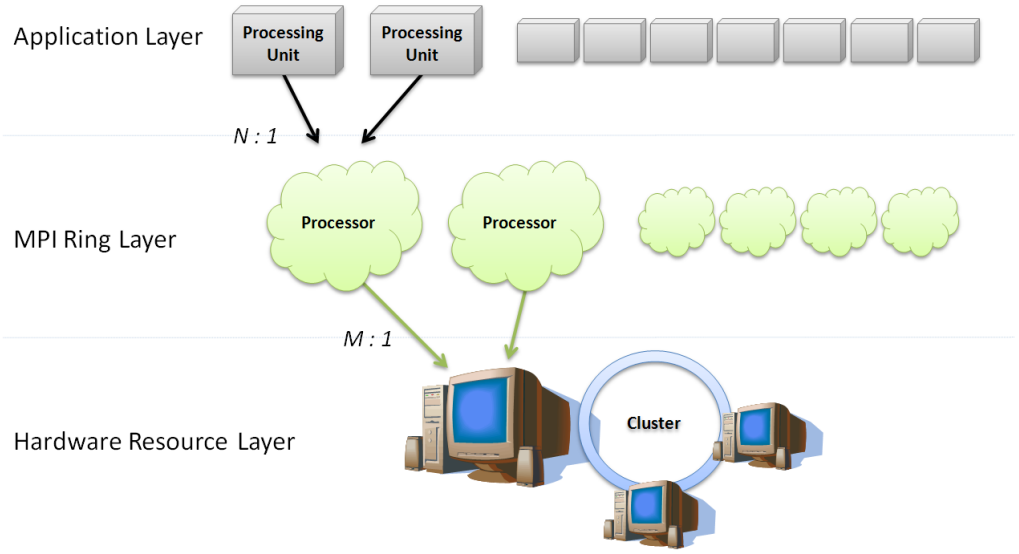

Figure B.9 *Layers of abstraction in the automatic parallelization and distribution framework.*

B.2.3 MPI Clusters

For the implementation of the second and third layer the *Message Passing Interface* [*MPI, A Message-Passing Interface Standard* 1995] is used as a middleware. MPI is the de facto standard for high-performance computing (HPC) environments and hence provides support for a large variety of platforms and has a huge user pool. MPI also supports a wide range of cluster participants, from very few or even a single computer to several hundreds or thousands. The standard allows for composition of computing environments by means of joining individuals to the so-called *MPI ring*. The MPI implementation then chooses the fastest channel for data exchange within the participants of the ring (be it e.g., shared memory, ethernet, infiniband, etc.).

Another advantage of MPI is, that extending or setting up an MPI ring is straight forward, as it only requires to start the MPI system service (daemon) on a computer. Also, at compile-time of a distributed application, it is not necessary to know which and how many computers actually form the MPI ring in the execution environment. So scaling an application is simple: the administrator only needs to add a new computer to the ring.

When adding a computing resource to an existing ring, the administrator specifies the number of processors for a resource. Note, that in the MPI terminology, a *processor* does not specify a CPU as such, but in fact a process started on a computer. The number of processors need not match the number of CPUs available

on a computer, but can be set to an arbitrary number. This allows for simulating cluster setups on a single computer - an essential mechanism for debugging. In a productive environment though, it makes sense to specify the number of processors according to the number of CPUs.

In a typical scenario, to exploit best the parallelization potentials of a hardware setup, one would on the one hand implement at least as many processing units, as MPI processors are available, and on the other hand one would set the number of processors on each MPI ring member at least to the number of CPUs available.

B.2.4 Distributed Processors and Synchronization

When an application built with the proposed framework is run, the user thus has two general options:

- *Direct Execution*
  The application is run by executing the compiled file directly. This is called local execution also. In this case the application automatically parallelizes on the local, currently used computer as described in Section B.2.1 according to the structural components and the number of CPUs available. It is equivalent to running the application on an MPI cluster with only one processor and a single ring participant (Figure B.10).

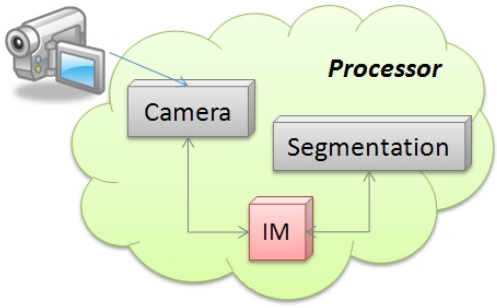

Figure B.10 *Processing units are bound to a processor at runtime, an information manager (IM) handles the communication.*

- *MPI Execution*
  The application is run using MPI tools, e.g., `mpiexec`. This triggers task distribution within the MPI ring. In this case, at boot time (see Section B.4 for a code snippet) the application environment automatically checks for presence of processors participating the MPI ring. Here, the application's processing units are assigned to an arbitrary processor (which is assigned to a ring mem-

ber) and thus PUs are automatically distributed to available MPI resources (see FIGURE B.11).

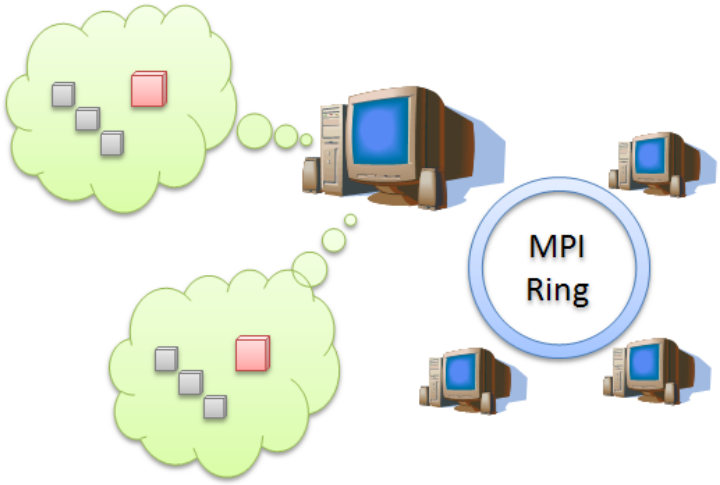

Figure B.11 *The MPI ring specifies the number of processors and binds them to physical hardware resources at runtime.*

Processors in the sense of MPI only carry a unique identifier, the so-called *rank* and no further information on the physical resource of a processor. The processor on which the application is started has rank zero. Within the proposed framework, processors also carry information about their host computer (e.g., the IP-address) and process information (for example the `pid`).

The concept for data synchronization and exchange between PUs implemented in the presented framework has already been introduced in SECTION B.1.2. There, it has been mentioned, that the information manager used for this task is implemented as a singleton for each application. However, considering a distributed scenario, it is not only necessary to synchronize data between processing units on the highest layer, but also between MPI processors one layer below. Thus, each MPI processor needs to be initialized with its own information manager for MPI layer data exchange. At runtime, the information managers then synchronize bi-directionally on-demand using MPI methodology[2] (see FIGURE B.12).

On-demand synchronization between information managers is possible, as the processing units define data and event channels that can be connected. For efficiency, only connected channels are to be synchronized in an application's runtime hard-

[2] An implementation detail: the MPI calls are not directly executed, but the powerful MPI wrapper utilities of *Boost* (`http://www.boost.org`) were used. This allows for convenient transmission of arbitrary datatypes given that they are serializable with Boost's serialization engine.

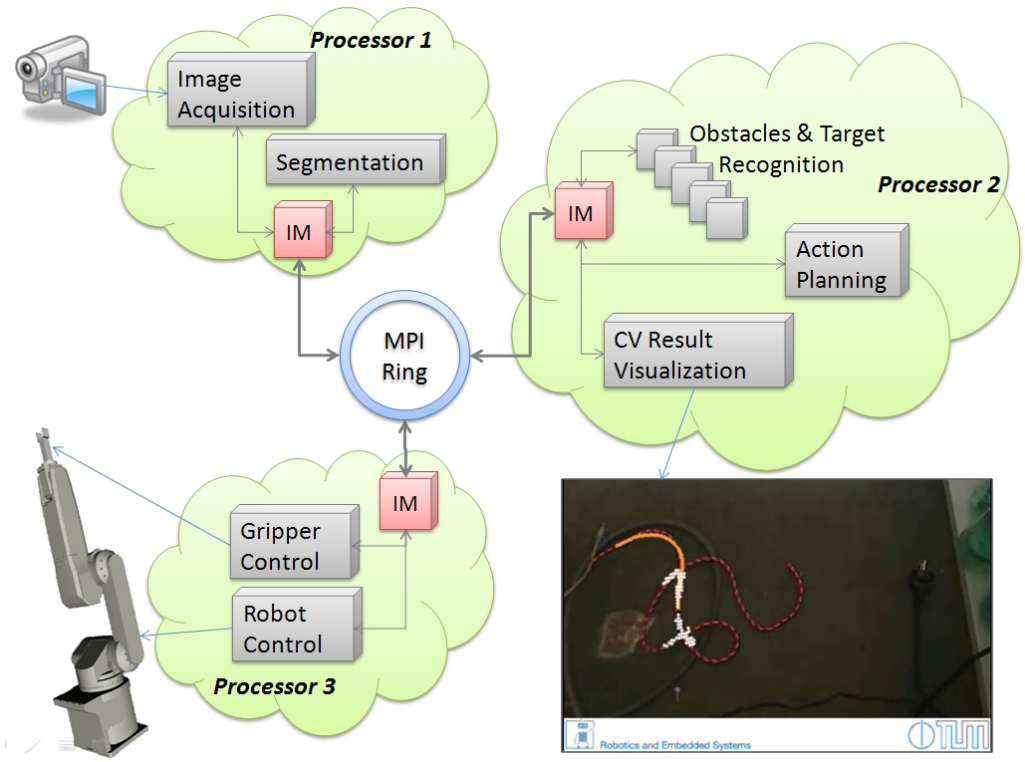

Figure B.12 *Information managers of each processor are synchronized via MPI. The figure shows one possible runtime layout for a distributed sensorimotor application.*

ware setup with multiple MPI processors. Even more, only when data is requested from the manager the first time (or in the case of a synchronous channel the first write operation on the channel occurs), the MPI synchronization is initialized on the information managers (i.e., the inter-processor level).

Unfortunately, as the most tasks implemented with the proposed work involve visual processing, the mechanisms described above do not work as well as expected for any runtime configuration. This has some obvious reasons: MPI and most of its implementations, e.g., `MPICH2`[3] are designed to transmit messages, i.e., preferentially text messages or small data chunks. Thus, transmission of for example strings, short integers or even matrices and vectors does works efficiently, i.e., reaches realtime performance easily on recent network and hardware infrastructures. But compared to matrices and the like, where serialized messages have the size of a few (hundred) bytes, a serialized image can easily reach a few million bytes (several megabytes). Thus, it is necessary to implement a special synchronization mechanism for large

[3] `http://www.mcs.anl.gov/research/projects/mpich2`

data (in particular image) transfer[4] between information managers of different processors. In practice, the sychronization for images is hence based on on-demand data streaming.

Whenever an information manager of a processor requires for example an image that is not only connected on the same processor, a bi-directional image stream is connected between the information managers of the processors hosting the connected processing units. The image streams are encoded and decoded on the fly using the `Xvid`-codec[5]. The drawback of this approach is the loss of image quality on a roundtrip. But in practice this is not of much relevance, as most applications do not implement extensive round-trips, but image processing units usually generate new, modified intermediate, or result images; or abstract result representations like object classifications or locations. The advantage is obvious: now it is possible to distribute visual processing to a cluster of computers which enables much higher performance and excellent scaling given that the application's structural components are well-defined. Furthermore, still the user does not have to deal with the parallelization across multiple computers explicitly, as this is handled automatically by the framework.

B.2.5 Explicit Physical Binding

In some cases it is useful, if not necessary, to have a possibility to specify at least the computer that a processing unit has to run on. For example, a camera device might be connected to a certain computer or the GUI needs to be displayed on some dedicated display.

To cope with this requirement, processing units provide a functionality to enforce a physical binding. Physical binding refers enforces a mapping from the processing unit (i.e., the application layer) to the hardware layer (physical resource layer). Whenever the physical binding is specified, the configured unit is only run on the requested machine. Inherently, this forces the application to terminate if no processor is found on the requested resource. By default though, the physical binding is left empty and the framework decides autonomously, where to run a unit.

To avoid loosing the generic runtime distribution facilities of the framework, usually the physical binding of a processing unit is specified within an application's configuration file. In this way only the configuration needs to be modified to reflect for example plugging in a camera device to a different MPI ring participant.

[4] To be exact: within the implementation a specialization of the general templated `Synchronizer`-class was introduced.

[5] `http://www.xvid.org`

B.3 Application Development and Deployment

The following sections briefly introduce the development and deployment architecture used in the framework. This architecture ensures convenient application development as well as integration of new modules into the core framework (see Section B.3.1). In Section B.3.2 an additional goal is to elaborate on methodology for automated building, testing, and deploying of the software implemented as part of the thesis.

B.3.1 Framework SDK

The framework is designed as a plug-in system based on shared libraries. This means the application developer has to link against the core framework library and my extend it with custom libraries. Then an application can be linked against all implemented libraries and in this way integrate modules developed earlier.

The reasoning behind this approach is the following. The framework core provides all necessary means for parallelization on multi-core machines and automatic distribution in cluster environments. The prototypes for processing units are also provided, including inter-unit communication facilities and event handling, the information managers for inter-machine communication and methods to retrieve environment information. This core functionality is encapsulated from the application developer, so the software developer only needs to implement the task logic into processing units, for example to integrate a custom hardware device, or introduce an application task workflow. The implemented processing units are packaged into a module library at varying granularity (at disposal of the software engineer) which can then be used in different applications yielding efficient code reusability.

The application itself then only loads the predefined processing units and defines the event and data channel connections. As pointed in Chapter 5, for the future it is intended to provide means to do so in a graphical user interface for easy plug-and-play application composition. Up to this point, a processing unit parameter configuration can be loaded from an `XML` definition file in order to restore a certain application state at start-up.

Finally, the build system of the framework produces an executable which can be used on a single PC not running any MPI components, as well as on a distributed system with multiple MPI ring members. The application decides at startup, which mode is to be loaded and how processing units are to be distributed on multi-core machines or cluster resources.

B.3.2 Build Virtualization and Deployment

The frameworks build system is designed to build and test developed applications on-the-fly, i.e., for example in a nightly build environment. For this purpose, scripts have been created in order to check out source code from a repository automatically, trigger a build process and create an installer. These processes also generate `HTML` report files.

For efficient automation of these processes certain dedicated virtual machines are set up. These virtual machines are powered up according to a scheduling algorithm on the host machine(s). The process is depicted schematically in Figure B.13.

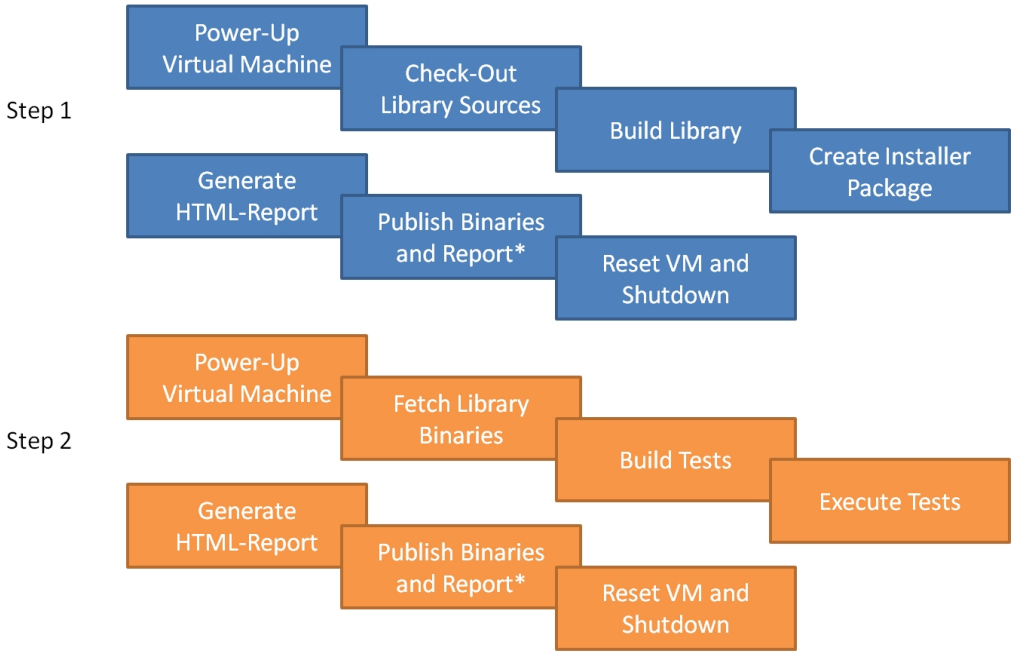

Figure B.13 *Automated build, test, and result-tracking process.*

The framework project, comprises of three major parts, (1) the library itself, (2) multi-layer tests and (3) tutorial applications. A brief overview of the requirements for the automated build, test and deploy framework are given in Table B.1.

As one can see from the table, the three parts have different dependencies, build specifications, deployment scenarios and access permissions. The results of the automated build process are published according to the access permissions to a web-service that provides the `HTML` reports, test and build log files and the binaries built in the process.

	Library	**Tests**	**Tutorials**
Source repository	Subversion	Subversion	Subversion, HTTP
Binary repository	Subversion, HTTP	Subversion	-
Code Variability	High	Medium	Low
Build Environment	Yes	Yes	No
Dependencies	3rd Party	3rd Party, Library	3rd Party, Library
Package Contents	Library binaries, Headers, Reference manual	-	Sources, Binaries, User guide
Error Reporting	Build, Packaging, Installation	Build, Execution	Build
Access Permissions	Developers, End-Users	Developers	Developers End-Users

Table B.1 *Framework overview*

The build system of the framework is in principle also able to accomplish tasks for cross platform target operating systems and various software configurations, as shown in the OpenTL library project, from which it is originally adapted (see details in [Müller and Knoll 2009b]). This can be achieved due to virtualization. Virtual operating systems are instantiated for each task in a non-persistent mode, so at start-up it can be guaranteed, that the virtual machine is in a "good" state. Thus the system produces repeatable results for equal sources on each of the target platforms.

Furthermore, based on the setup of the virtual machines, it is quite easy to exactly specify the requirements of physical hardware for the end user and investigate the performance difference emerging from higher memory capacity, better CPU availability, etc. For this purpose one simply has to reconfigure the virtual machine, which facilitates the evaluation process enormously. Finally, it is also possible to replace the host system conveniently, as the virtual machine images can simply be copied to the new host and run directly. All the new host needs to do is provide a public folder for the results and take this folder as the document root for a webserver.

B.4 System Manual

The following is intended to give a short introduction on how to install the core framework and necessary third-party dependencies, and furthermore how to implement a simple distributed application. All steps can be reviewed in greater detail from the tutorial[6]. The following sections refer to an installation on *Ubuntu*[7].

B.4.1 Third-Party Dependencies

As Ubuntu is entirely open-source, all third-party dependencies can be downloaded and installed directly from corresponding repositories.

```
sudo aptitude install build-essential ccache cmake \
cmake-curses-gui libcv4 libcv-dev libcvaux4 libcvaux-dev \
libhighgui4 libhighgui-dev libqt4-dev libqt4-dbg libsoqt4-dev \
libcoin60-dev libboost-all-dev libboost-mpi-dev libmpich2-dev \
eclipse libspnav-dev spacenavd subversion
```

Downloading and installing all these packages can take several minutes, depending on available hardware and internet connection.

B.4.2 Eclipse - the IDE

It is recommended to install three more or less essential (but anyway useful) extensions to *Eclipse*[8]. To do so, the following update sites have to be added:

```
# C/C++ Development Tooling (CDT)
http://download.eclipse.org/releases/galileo
# CMakeEd
http://cmakeed.sourceforge.net/eclipse/
# Subclipse
http://subclipse.tigris.org/update_1.6.x/
```

B.4.3 Setting up MPI

Create the MPI configuration file in a user home directory:

```
cd $HOME
touch .mpd.conf
```

[6] http://www.flexiblerobotics.com
[7] http://www.ubuntu.com
[8] http://www.eclipse.org

```
echo MPD_SECRETWORD=[mysecretword] > .mpd.conf
chmod 600 .mpd.conf
```

Start the MPI-daemon on the master:

```
mpd --listenport=[master-port]
```

Start it on the slaves (make other computers join the ring):

```
mpd --host=[master-ip] --port=[master-port] \
    --ifhn=[slave-ip] --listenport=[slave-port]
```

Check for ring-members with (on any of the computers):

```
mpdtrace -l
```

Start an MPI-application (here: passing local variable `LD_LIBRARY_PATH`):

```
mpiexec -genvlist LD_LIBRARY_PATH [executable] [commandline-arg]*
```

B.4.4 Building an Application

This is a short snippet for implementing a (very) simple application to displaying the live image of a standard OpenCV-camera using the graphical user interface (GUI) unit.

```
#include <frf.h>

int main(int argc, char **argv)
{
  // Boot distributed environment
  frf::boot(argc, argv);

  // Setup processing units
  frf::vision::camera::OpenCVCamera* cam = new frf::vision::camera::
      OpenCVCamera();
  frf::gui::GuiUnit* gui = new frf::gui::GuiUnit();

  // Initialize the processing units
  frf::init();

  // Connect the shared parameters
```

```
16   unsigned int id;
17   id = cam->share<frf::core::data::Image>(frf::vision::camera::
         OpenCVCamera::LIVE_IMAGE);
18   gui->connectTo(id, "Live Images",
19                  new frf::gui::feedback::Image("USB Image"));
20
21   // Run distributed application
22   return frf::run();
23 }
```

Note that start events are automatically fired to all units at `frf:run()` if not specified explicitly. The stop events are on the other hand automatically connected to the GUI per default. FIGURE B.14 shows a screenshot of the running application.

Figure B.14 *Screenshot of the simple application described in the text.*

Publications

Foster, M. E., Giuliani, M., Müller, T., Rickert, M., Knoll, A., Erlhagen, W., Bicho, E., Hipólito, N. and Louro, L.: 2008, Combining goal inference and natural-language dialogue for human-robot joint action, *Proceedings of the International Workshop on Combinations of Intelligent Methods and Applications, European Conference on Artificial Intelligence*.

Giuliani, M., Lenz, C., Müller, T., Rickert, M. and Knoll, A.: 2010, Design principles for safety in human-robot interaction, *International Journal of Social Robotics* **2**(3), 253–274.

Müller, T.: 2006, Objekterkennung und Klassifikation für JAST - Joint Action Science and Technology. Systementwicklungsprojekt, Technische Universität München.

Müller, T.: 2007, Konzeption und Implementierung eines adaptiven, robusten und effizienten Verfahrens zur Klassifikation von Objekten mittels Template Matching. Diplomarbeit, Technische Universität München.

Müller, T. and Knoll, A.: 2008a, Bioinspired early visual processing: The attention condensation mechanism, *Proceedings of the Australasian Conference on Robotics and Automation*.

Müller, T. and Knoll, A.: 2008b, Humanoid early visual processing using attention mechanisms, *Proceedings of Cognitive Humanoid Vision WS at IEEE-RAS International Conference on Humanoid Robots*.

Müller, T. and Knoll, A.: 2009a, Attention driven visual processing for an interactive dialog robot, *Proceedings of the 24th ACM Symposium on Applied Computing*.

Müller, T. and Knoll, A.: 2009b, Virtualization techniques for cross platform automated software builds, tests and deployment, *Proceedings of the Fourth International Conference on Software Engineering Advances*.

Müller, T. and Knoll, A.: 2010, A generic approach to realtime robot control and parallel processing for industrial scenarios, *Proceedings of the 7th IEEE International Conference on Industrial Technology*.

Müller, T. and Knoll, A.: 2011, Evolution of attention mechanisms for early visual processing, *Proceedings of IS&T / SPIE Electronic Imaging*.

Müller, T., Lenz, C., Barner, S. and Knoll, A.: 2008, Accelerating integral histograms using an adaptive approach, *Proceedings of the 3rd International Conference on Image and Signal Processing*, Lecture Notes in Computer Science (LNCS), Springer, pp. 209–217.

Müller, T., Tran, B. A. and Knoll, A.: 2010, A flexible robotics and automation system: Parallel visual processing, realtime actuator control, and task automation for limp object handling, *Proceedings of the 7th International Conference on Informatics in Control, Automation and Robotics*.

Müller, T., Tran, B. A. and Knoll, A.: 2011, Automatic distribution of vision-tasks on computing clusters, *Proceedings of IS&T / SPIE Electronic Imaging*.

Müller, T., Ziaie, P. and Knoll, A.: 2008, A wait-free realtime system for optimal distribution of vision tasks on multicore architectures, *Proceedings of the 5th International Conference on Informatics in Control, Automation and Robotics*.

Radi, M., Reiter, A., Zäh, M. F., Müller, T. and Knoll, A.: 2010, Telepresence technology for production: From manual to automated assembly, *Proceedings of EUROHAPTICS 2010*.

Tuong, N. X., Müller, T. and Knoll, A.: 2011, Robust pedestrian detection and tracking from a moving vehicle, *Proceedings of IS&T / SPIE Electronic Imaging*.

Ziaie, P., Müller, T., Foster, M. E. and Knoll, A.: 2008, A naïve Bayes classifier with distance weighting for hand-gesture recognition, *Proceedings of the 13th International CSI Computer Conference*.

Ziaie, P., Müller, T., Foster, M. E. and Knoll, A.: 2009, A naïve Bayes classifier with distance weighting for hand-gesture recognition, *Advances in Computer Science and Engineering*, Vol. 6 of *Communications in Computer and Information Science*, Springer Berlin Heidelberg, pp. 308–315.

Ziaie, P., Müller, T. and Knoll, A.: 2008, A novel approach to hand-gesture recognition in a human-robot dialog system, *Proceedings of the First Intl. Workshop on Image Processing Theory, Tools & Applications*.

References

Agrawal, M.: 2005, A lie algebraic approach for consistent pose registration for general euclidean motion, *Intl. Conference on Intelligent Robots and Systems*.

Anderson, M. L.: 2003, Embodied cognition: A field guide, *Artificial Intelligence* **149**.

Arkin, R. C.: 1998, *Behavior-Based Robotics*, MIT Press.

Bach-y-Rita, P.: 1967, Sensory plasticity: Applications to a vision substitution system, *Acta Neurologica Scandinavica* **43**(4), 417–426.

Balkenius, C. and Hulth, N.: 1999, Attention as selection-for-action: a scheme for active perception, *Advanced Mobile Robots, 1999. (Eurobot '99) 1999 Third European Workshop on*, pp. 113–119.

Ball, R. S.: 1876, *The theory of screws: A study in the dynamics of a rigid body*, Hodges, Foster.

Bay, H., Ess, A., Tuytelaars, T. and Gool, L. V.: 2008, SURF: Speeded up robust features, *Computer Vision and Image Understanding* **110**(3), 346–359.

Blaschke, W.: 1960, *Kinematik und Quaternionen*, VEB Deutscher Verlag der Wissenschaften.

Boudreau, R. and Podhorodeski, R. P.: 2010, Singularity analysis of a kinematically simple class of 7-jointed revolute manipulators, *Transactions of the Canadian Society for Mechanical Engineering* **34**(1).

Bouguet, J.-Y.: 2000, Pyramidal implementation of the Lucas Kanade feature tracker, *Technical report*, Intel Corporation, Microprocessor Research Labs.

Bradski, G. R.: 1998, Computer vision face tracking for use in a perceptual user interface, *Intel Technology Journal* **Q2**.

Broadbent, D. E.: 1958, *Perception and communication*, Pergamon Press, London.

Brockett, R. W.: 1984, Robotic manipulators and the product of exponentials formula, *Mathematical Theory of Networks and Systems* **58**, 120–129.

Brooks, R. A.: 1986, A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation* **2**(1), 14–23.

Brooks, R. A.: 1991, Intelligence without representation, *Artificial Intelligence* **47**, 139–159.

Buss, S. R.: 2009, Introduction to inverse kinematics with jacobian transpose, pseudoinverse, and damped least squares methods, *Technical report*, University of California, San Diego, California.

Chapman, B., Jost, G. and van der Pas, R.: 2007, *Using OpenMP*, MIT Press.

Chaumette, F. and Hutchinson, S.: 2006, Visual servo control part i: Basic approaches, *IEEE Robotics & Automation Magazine* **13**(4), 82–90.

Chaumette, F. and Hutchinson, S.: 2007, Visual servo control part ii: Advanced approaches, *IEEE Robotics & Automation Magazine* **14**(1), 109–118.

Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L. E. and Thrun, S.: 2005, *Principles of Robot Motion: Theory, Algorithms, and Implementations*, The MIT Press.

Clark, A.: 1997, *Being There: Putting Brain, Body, and World Together Again*, The MIT Press.

Clifford, W. K.: 1873, Preliminary sketch of bi-quaternions, *Proc. London Math. Society*, Vol. 4, pp. 381–395.

Cohen, J. D., Aston-Jones, G. and Gilzenrat, M. S.: 2004, A systems-level perspective on attention and cognitive control: Guided activation, adaptive gating, conflict monitoring, and exploitation versus exploration, *Cognitive Neuroscience of Attention*, Guilford Publications, pp. 71–90.

Comaniciu, D., Ramesh, V. and Meer, P.: 2000, Real-time tracking of non-rigid objects using mean shift, *Computer Vision and Pattern Recognition*.

Corbetta, M.: 1998, Frontoparietal cortical networks for directing attention and the eye to visual locations: identical, independent, or overlapping neural systems?, *Proceedings of the National Academy of Sciences of the USA* **95**(3).

Corbetta, M. and Shulman, G.: 2002, Control of goal-directed and stimulus-driven attention in the brain, *Nature Reviews Neuroscience* **3**, 201–215.

Craig, J. J.: 1955, *Introduction to Robotics: Mechanics and Control*, Prentice Hall.

Dalal, N. and Triggs, B.: 2005, Histograms of oriented gradients for human detection, *Computer Vision and Pattern Recognition*, pp. 886–893.

Dayan, P. and Abbott, L.: 2001, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*, The MIT Press.

de Groote, H. F.: 1975, On the computational complexity of quaternion multiplication, *Information Processing Letters* (6), 177–179.

de Michieli, L., Nori, F., Pini Prato, A. and Sandini, G.: 2008, Study on humanoid robot systems: an energy approach, *IEEE-RAS International Conference on Humanoid Robots*.

Denavit, J. and Hartenberg, R. S.: 1955, A kinematic notation for lower-pair mechanisms based on matrices, *Trans ASME J. Appl. Mech* pp. 215–221.

Dobkin, D.: 1973, *On the Arithmetic Complexity of a Class of Arithmetic Computations*, PhD thesis, Harvarad University.

Drummond, T. and Cipolla, R.: 1999a, Real-time tracking of complex structures with on-line camera calibration, *In Proc. British Machine Vision Conference (BMVC'99*, pp. 574–583.

Drummond, T. and Cipolla, R.: 1999b, Visual tracking and control using lie algebras, *In Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, IEEE Computer Society Press, pp. 652–657.

Drummond, T. and Cipolla, R.: 2000, Application of lie algebras to visual servoing, *International Journal of COmputer Vision* **37**(1), 21–41.

Drummond, T. and Cipolla, R.: 2002, Real-time visual tracking of complex structures, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **24**(7), 932–946.

Dynkin, E. B.: 1947, Calculation of the coefficients in the campbell–hausdorff formula, *Proceedings of the USSR Academy of Sciences*, Vol. 57, pp. 323–326.

Eberly, D.: 2011, Euler angle formulas, *Technical report*, Geometric Tools, LLC. http://www.geometrictools.com.

Echtle, K.: 1998, *Fehlertoleranzverfahren*, Springer-Verlag GmbH.

Eskridge, B. E. and Hougen, D. F.: 2009, Using action abstraction to evolve effective controllers, *Genetic and Evolutionary Computation Conference*.

Euler, L.: 1776, *Novi Commentarii academiae scientiarum Petropolitanae*, Vol. 20, St. Petersburg Academy.

Flash, T. and Hogan, N.: 1985, The coordination of arm movements: An experimentally confirmed mathematical model, *Journal of Neuroscience* **5**(7), 1688–1703.

Fletcher, P. T., Lu, C. and Joshi, S.: 2003, Statistics of shape via principal geodesic analysis on lie groups, *Intl. Conference on Computer Vision and Pattern Recognition*.

Foster, M. E., By, T., Rickert, M. and Knoll, A.: 2006, Humanrobot dialogue for joint construction tasks, Proc. ICMI.

Gallese, V. and Lakoff, G.: 2005, The brain's concepts: The role of the sensory-motor system in conceptual knowledge, *Cognitive Neuropsychology* **21**(0).

Geib, C., Mourao, K., Petrick, R., Pugeault, N., Steedman, M., Krueger, N. and Wörgötter, F.: 2006, Object action complexes as an interface for planning and robot control, *IEEE-RAS International Conference on Humanoid Robots*.

Gilles, S.: 1998, *Robust Description and Matching of Images*, PhD thesis, University of Oxford.

Giuliani, M.: 2011, *Multimodal Fusion for Human-Robot Interaction*, PhD thesis, Technische Universität München.

Govindu, V. M.: 2003, Lie-algebraic averaging for globally consistent motion estimation, *Intl. Conference on Computer Vision and Pattern Recognition*.

Grassia, F. S.: 1998, Practical parameterization of rotations using the exponential map, *Journal of Graphics Tools* **3**, 29–48.

Graves, A. R. and Czarnecki, C.: 2000, Design patterns for behavior-based robotics, *IEEE Transactions On Systems, Man, And Cybernetics* **30**(1).

Hamilton, W. R.: 1853, *Lectures on Quaternions*, Royal Irish Academy.

Hara, F. and Pfeifer, R.: 2000, On the relation among morphology, material and control in morpho-functional machines, *Sixth International Conference on the Simulation of Adaptive Behavior*, MIT Press, pp. 33–40.

Hawkins, J.: 2004, *On Intelligence*, Times Books.

Hebb, D. O.: 1949, *The Organization of Behavior: A Neuropsychological Theory*, Wiley and Sons, New York.

Hellerer, M.: 2009, Potential field based position control for Mitsubishi RV-6S industrial robots. Bachelor's Thesis, Department of Informatics, Technische Universität München.

Higham, N.: 2004, The scaling and squaring method for the matrix exponential revisited, *SIAM J. Matrix Anal. Appl.* **26**(4), 1179–1193.

Hinton, G. E.: 2007, To recognize shapes, first learn to generate images, *in* P. Cisek, T. Drew and J. Kalaska (eds), *Computational Neuroscience: Theoretical Insights into Brain Function*, Elsevier.

Hogan, N.: 1984, Adaptive control of mechanical impedance by coactivation of antagonist muscles, *IEEE Transactions on Automatic Control* **AC-29**(8), 681–690.

Hollerbach, J. M.: 1985, Optimum kinematic design for a seven degree of freedom manipulator, *Robotics Research: The Second International Symposium*, MIT Press, pp. 215–222.

Hommel, B.: 2010, Grounding attention in action control: The intentional control of selection, *in* B. Bruya (ed.), *Effortless attention: A new perspective in the cognitive science of attention and action*, The MIT Press, pp. 121–140.

Howell, T. D. and Lafon, J.-C.: 1975, The complexity of the quaternion product, *Technical report*, Cronell University, Ithaca, N.Y.

Huber, M., Radrich, H., Wendt, C., Rickert, M., Knoll, A., Brandt, T. and Glasauer, S.: 2009, Evaluation of a novel biologically inspired trajectory generator in human-robot interaction, *The 18th IEEE International Symposium on Robot and Human Interactive Communication*, pp. 639 – 644.

Ishiguro, A. and Kawakatsu, T.: 2003, How should control and body systems be coupled? a robotic case study., *Embodied Artificial Intelligence'03*, pp. 107–118.

Itti, L. and Koch, C.: 2001, Computational modelling of visual attention, *Nature Reviews Neuroscience* **2**, 194–203.

Itti, L., Koch, C. and Niebur, E.: 1998, A model of saliency-based visual attention for rapid scene analysis, *Pattern Analysis and Machine Intelligence* **20**(11), 1254–1259.

Jackson, D.: 1942, The instantaneous motion of a rigid body, *The American Mathematical Monthly* **49**(10), 661–667.

James, W.: 1890, *Principles of Psychology*, Harvard University Press.

Johnson, M. and Demiris, Y.: 2005, Hierarchies of coupled inverse and forward models for abstraction in robot action planning, recognition and imitation, *Social Intelligence and Interaction in Animals, Robots and Agents*.

Joukhadar, A. and Laugier, C.: 1997, Dynamic simulation: Model, basic algorithms, and optimization, *In Proc. of the Workshop on the Algorithmic Foundations of Robotics, Toulouse (FR*, A.K. Peters Publisher, pp. 419–434.

Kadir, T. and Brady, M.: 2000, Saliency, scale and image description, *IJCV* .

Kahneman, D.: 1973, *Attention and Effort*, Prentice-Hall, Inc.

Kalman, R. E.: 1960, A New Approach to Linear Filtering and Prediction Problems, *Journal of Basic Engineering* **82**(1), 35–45.

Kavan, L., Collins, S., O'Sullivan, C. and Zara, J.: 2006, Dual quaternions for rigid transformation blending, *Technical report*, Trinity College Dublin.

Khatib, O.: 1985, The potential field approach and operational space formulation in robot control, *Proceedings of the Fourth Yale Workshop on Applications of Adaptive Systems Theory*, Yale University.

Khatib, O.: 1987, A unified approach for motion and force control of robot manipuators: The operational space formulation, *IEEE Journal of Robotics and Automation* **RA-3**(1).

Kienzle, W., Wichmann, F. A., Schölkopf, B. and Franz, M. O.: 2006, A nonparametric approach to bottom-up visual saliency, *Proc. NIPS '06*.

Krüger, N., Piater, J., Wörgötter, F., Geib, C., Petrick, R., Steedman, M., Ude, A., Asfour, T., Kraft, D., Omrcen, D., Hommel, B., Agostini, A., Kragic, D., Eklundh, J.-O., Krüger, V., Torras, C. and Dillmann, R.: 2009, A formal definition of object-action complexes and examples at different levels of the processing hierarchy, *Technical report*, EU project PACO-PLUS.

Laaksonen, J., Felip, J., Morales, A. and Kyrki, V.: 2010, Embodiment independent manipulation through action abstraction, *IEEE International Conference on Robotics and Automation*.

Latombe, J. C.: 1991, *Robot Motion Planning*, Kluwer Academic Publishers.

Laumond, J.-P. (ed.): 1998, *Robot Motion Planning and Control*, Vol. 229 of *Lectures Notes in Control and Information Sciences*, Springer.

Lavie, N., Hirst, A., de Fockert, J. W. and Viding, E.: 2004, Load theory of selective attention and cognitive control, *Journal of Experimental Psychology* **133**(3), 339–354.

Lawson, J. D.: 1967, Generalized Runge-Kutta processes for stable systems with large Lipschitz constants, *SIAM J. Num. Anal.* **4**, 372–380.

Lee, B.-U.: 1991, *Stereo Matching of Skull Landmarks*, PhD thesis, Stanford University.

Lepetit, V. and Fua, P.: 2005, Monocular model-based 3d tracking of rigid objects, *Foundations and Trends in Computer Graphics and Vision* **1**(1), 1–89.

Levenberg, K.: 1944, A method for the solution of certain non-linear problems in least squares, *The Quarterly of Applied Mathematics* **2**, 164–168.

Lie, M. S.: 1872, *On a Class of Geometric Transformations*, PhD thesis, University of Christiania.

Lienhart, R. and Maydt, J.: 2002, An extended set of haar-like features for rapid object detection, *IEEE ICIP 2002*, pp. 900–903.

Loock, A. and Schömer, E.: 2001, A virtual environment for interactive assembly

simulation: From rigid bodies to deformable cables, *5th World Multiconference on Systemics, Cybernetics and Informatics (SCI01)*, Vol. 3, pp. 325–332.

Lucas, B. D.: 1984, *Generalized Image Matching by the Method of Differences*, PhD thesis, Robotics Institute, Carnegie Mellon University.

Lucas, B. D. and Kanade, T.: 1981, An iterative image registration technique with an application to stereo vision, *Imaging Understanding Workshop*, pp. 121–130.

Manocha, D. and Canny, J. F.: 1994, Efficient inverse kinematics for general 6r manipulators, *IEEE Transactions on Robotics and Automation* **10**, 648–657.

Mayer, H., Nagy, I., Knoll, A., Braun, E. U. and Bauernschmitt, R.: 2007, Adaptive control for human-robot skill transfer: Trajectory planning based on fluid dynamics, *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1800–1807.

Meyberg, K. and Vachenauer, P.: 2001, *Höhere Mathematik 1*, 6 edn, Springer.

Moler, C. B. and Van Loan, C. F.: 1978, Nineteen dubious ways to compute the exponential of a matrix, *SIAM Rev.* **20**, 801–036.

Moler, C. B. and Van Loan, C. F.: 2003, Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later, *SIAM Rev.* **45**, 3–49.

Mountcastle, V. B.: 1978, An organizing principle for cerebral functions, *The Mindful Brain*, MIT Press.

MPI, A Message-Passing Interface Standard: 1995, *Technical report*, University of Tennessee, Knoxville, Tennessee.

Nelder, J. and Mead, R.: 1965, A simplex method for function minimization, *Computer Journal* **7**, 308–313.

Newman, S. D., Keller, T. A. and Just, M. A.: 2007, Volitional control of attention and brain activation in dual task performance, *Human Brain Mapping* **28**, 109–117.

Nguyen, K. D., Ng, T.-C. and Chen, I.-M.: 2008, On algorithms for planning s-curve motion profiles, *Internation Journal of Advanced Robotic Systems* **5**(1), 99–106.

Noë, A.: 2004, *Action in Perception*, The MIT Press.

O'Regan, J. K. and Noë, A.: 2001, A sensorimotor account of vision and visual consciousness, *Behavioural and Brain Sciences* **24**, 939–1031.

Panin, G., Lenz, C., Nair, S., Roth, E., in Wojtczyk, M., Friedlhuber, T. and Knoll, A.: 2008, A Unifying Software Architecture for Model-based Visual Tracking, *Proc. of the 20th Annual Symposium of Electronic Imaging*.

Pfeifer, R.: 2000, On the role of morphology and materials in adaptive behavior, *Sixth International Conference on the Simulation of Adaptive Behavior*, MIT Press, pp. 23–32.

Pfeifer, R. and Bongard, J.: 2005, *How the Body Shapes the Way We Think*, MIT Press.

Pfeifer, R. and Gómez, G.: 2005, Interacting with the real world: design principles for intelligent systems, *Artificial Life and Robotics* **9**(1), 1–6.

Pfeifer, R. and Iida, F.: 2004, Embodied artificial intelligence: Trends and challenges, *in* F. Iida, R. Pfeifer, L. Steels and Y. Kuniyoshi (eds), *Embodied Artificial Intelligence*, Vol. 3139 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 629–629.

Plopski, A.: 2010, Entwicklung einer markerbasierten visuellen Kontrollschnittstelle für insdustrielle Roboter. Bachelor's Thesis, Department of Informatics, Technische Universität München.

Posner, M. and Cohen, Y.: 1984, Components of visual orienting, *Attention and Performance* **10**, 531–556.

Pozzo, T., McIntyre, J., Cheron, G. and Papaxanthis, C.: 1998, Hand trajectory formation during whole body reaching movements in man, *Neuroscience Letters* **240**, 159 – 162.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P.: 2002, *Numerical Recipes in C++*, 2nd edn, Cambridge University Press.

Provot, X.: 1995, Deformation constraints in a mass-spring model to describe rigid cloth behavior, *in* W. A. Davis and P. Prusinkiewicz (eds), *Graphics Interface '95*, Canadian Human-Computer Communications Society, pp. 147–154.

Reinagel, P. and Zador, A. M.: 1999, Natural scene statistics at the center of gaze, *Network: Computation in Neural Systems* **10**(4), 341–350.

Reuleaux, F.: 1875, *Theoretische Kinematic: Grundzüge einer Theorie des Maschinwesens*, Vieweg, Braunschweig.

Rickert, M., Brock, O. and Knoll, A.: 2008, Balancing Exploration and Exploitation in Motion Planning, *Proc. of the IEEE International Conference on Robotics and Automation*.

Rickert, M., Foster, M. E., Giuliani, M., By, T., Panin, G. and Knoll, A.: 2007, Integrating language, vision, and action for human robot dialog systems, Proc. ICMI.

Rudin, W.: 1986, *Real and Complex Analysis*, 3 edn, McGraw-Hill.

Samelson, H.: 1990, *Notes on Lie Algebras*, 2nd edition edn, Springer.

Sciavicco, L. and Siciliano, B.: 2001, *Modelling and control of robot manipulators*, Springer, Berlin.

Selig, J. M.: 2003, Lie Groups and Lie Algebras in Robotics, *Computational Noncommutative Algebra and Applications*, Springer Netherlands, pp. 101–125.

Selig, J. M.: 2005, *Geometric Fundamentals of Robotics*, 2nd edn, Springer.

Shoemake, K.: 1985, Animating rotation with quaternion curves, *ACM SIGGRAPH '85* **19**(3).

Study, E.: 1891, Von den bewegungen und umlegungen, *Mathematische Annalen* **39**, 441–565. 10.1007/BF01199824.
URL: *http://dx.doi.org/10.1007/BF01199824*

Thomas, M. B.: 2003, *Advanced servo control of a pneumatic actuator*, PhD thesis, Ohio State University.

Tipper, C. and Kingstone, A.: 2005, Is inhibition of return a reflexive effect?, *Cognition* **97**, B55–B62.

Tran, B. A.: 2010, Detektion, Tracking und Robotergestützte Handhabung Linearer Deformierbarer Objekte. Master's Thesis, Department of Electrical Engineering, Technische Universität München.

Tuong, N. X.: 2010, Real-time pedestrian detection and tracking on moving vehicle. Thesis, Department of Computer Engineering, Nanyang Technological University (TUM exchange program), Singapore.

Varadarajan, V. S.: 1984, *Lie Groups, Lie Algebras, and Their Representation*, Vol. 102 of *Graduate Texts in Mathematics*, Springer.

Vicci, L.: 2001, Quaternions and rotations in 3-space: The algebra and its geometric interpretation, *Technical report*, University of North Carolina.

Viola, P. and Jones, M.: 2001, Robust real-time object detection, *IEEE ICCV Workshop on Statistical and Computational Theories of Vision*.

Völk, K.: 2010, Gait improvement of a humanoid robot with reinforcement learning. Bachelor's Thesis, Department of Informatics, Technische Universität München.

Walther, D., Itti, L., Riesenhuber, M., Poggio, T. and Koch, C.: 2002, Attentional selection for object recognition – a gentle way, *Biologically Motivated Computer Vision*, pp. 251–267.

Weyl, H.: 1939, *The Classical Groups: Their Invariants and Representations*, Princeton Press.

Wörgötter, F., Agostini, A., Krüger, N., Shylo, N. and Porr, B.: 2009, Cognitive agents – a procedural perspective relying on the predictability of object-action-complexes (oacs), *Robotics and Autonomous Systems* **57**(4), 420–432.

Wright, R. D. and Ward, L. M.: 2008, *Orienting of Attention*, Oxford University Press.