ICT Call 7
ROBOHOW.COG
FP7-ICT-288533

Deliverable D3.2:

Cognitive constraints for arm and hand manipulation

February 12th, 2014

Project acronym:	ROBOHOW.COG
Project full title:	Web-enabled and Experience-based Cognitive Robots that Learn Complex Everyday Manipulation Tasks
Work Package:	WP 3
Document number:	D3.2
Document title:	Cognitive constraints for arm and hand manipulation
Version:	2.0
Delivery date:	February 12th, 2014
Nature:	Report
Dissemination level:	Public (PU)
Authors:	Herman Bruyninckx (KU Leuven) Gianni Borghesan (KU Leuven) Moritz Tenorth (UniHB) Michael Beetz (UniHB)

The research leading to these results has received funding from the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement n°288533 ROBOHOW.COG.

Contents

Summary **4**

1 System architecture methodology **8**
1.1 D3.2 in the context of the RoboHow system architecture 8
1.2 D3.2 in relationship to Work Package 7 9
1.3 Composite Component software pattern 9
1.4 The "Hierarchical Hypergraphs" meta model 13
1.5 Integration of knowledge and learning 13
1.6 Integration of "legacy" components 15

2 Overview Year 2 R&D results **17**
2.1 Hand and Arm Control 17
2.2 Planning 18

3 Forewords to attached publications **19**
3.1 Control and constraint specification 19
3.2 Cognitive planning in the task space 21

Summary

This Deliverable summarizes the research efforts of the consortium in the field of smart ("cognitive") strategies for manipulation tasks. With the term "cognitive", we refer to the capability of a robotic system to connect the *motion* of its mechanical hardware to all sorts of *knowledge* about the purpose of that motion. For example, when the robot *opens a door*, its impedance behaviour must match the geometric and dynamic properties of a particular door; tasks like *slicing vegetables* require impedance control parameters that vary wildly over the different combinations of vegetables, tools and robots.
The presented research gives the consortium's constructive approaches that give robots access to formal and computer-implemented representations, reasoning rules and learning algorithms, allowing them to realise this connection between "motion" and "knowledge", in the following ways:

- *to configure* the models and their parameters, that determine a motion's desired pose, required speed and impedance, interpretation of "errors", etc., in the particular context of a particular task and robot system. System and task developers should be supported by a methodology that helps them introduce the knowledge that the robot should use during execution: what kind of models are relevant? with what set of model parameters? and how do these depend on the execution context?

- *to learn* such models and parameters from series of executions of the same task.

The contributions presented in this document are of two, complementary, types:

- *"behaviour"*: *how* (i.e., with which functionalities and algorithms) is the above-mentioned knowledge-driven configuration performed, or *how* can particular learning approaches fill in the models and their parameters?

- *"structure"*: what is the *architecture* (or "system composition", or "component interconnection") in which all these behaviours can be embedded such that the context of their relevance and validity is provided to the system developers in a methodological way, allowing them to define clearly the *scopes* within the system's software where each of the mentioned behaviours can be plugged in, and to which they can constrain their development efforts.

In Year 2, the partners involved in this Work Package have mainly focused on novel behaviour development, while KU Leuven has been driving the system composition developments.
This work is achieved within the Work Package's context of *"Constraint- and Optimization-based Control"*, and its position with respect to the Year 1 research presented in Deliverable D3.1 *"Documentation about the constraint-based framework"* is as follows:

- the methodological driver behind both is that *"everything is a constrained optimization problem"*.
- D3.1 used that approach to find the optimal motion for a complex robot system, *given* the formulation of the constrained optimization problem to be solved. The outcome are generic "solvers" that can make the robot *adapt* its motion to changes in all the *parameters* in the optimization problem that its sensors can *measure* at runtime.
- this Deliverable D3.2 extends the scope of the optimization to adaptation of the *formulation* of the optimization problem (including the *objective functions* and *constraints*, and not just the measurable parameters!) by means of "queries" to knowledge bases that contain the knowledge about which constrained optimization problem is the best one for the robot to "solve" at each instant in its mission execution.
- this step from "adaptation" to "cognition" means, roughly, the following, for the constrained optimization solvers in the system:
  - the adaptation case solves a system involving (i) *continuous* time and space relationships reflecting the robot's motion capabilities and its motion objectives, and (ii) *discrete* switches in the system of relationships to be solved.
  - the cognitive case adds (iii) *symbolic* knowledge relationships.

  Constrained optimization in the latter case if better known under the name of *"constraint satisfaction"*, but the concept is exactly the same as in the former case.
- from the *system architecture* point of view, bringing in "cognition" introduces tremendous changes in the complexity:
  - the system architecture of the adaptation case is completely driven by the "natural hierarchies" of the robots' mechanical hardware and motion control architecture: from power convertors, to actuators, to mechanical transmissions, to kinematic joints, and eventually to the whole kinematic chain of the robot.

    Hence, the system architecture complexity is rather limited.
  - the system architecture of the cognitive case extends the "hierarchical" structure with non-strict hierarchical compositions of "knowledge contexts", that is, the different sources of knowledge act on different parts of the motion control architecture.

    Since these sources of knowledge interact with the motion control architecture in overlapping and time-varying ways, the system architecture becomes more complex, and more application specific.

Hence, the bulk of the "generic integration" work in this Deliverable was spent on developing an innovative methodology to create such cognitive architectures, with the core primitive in this context being the so-called *Component Composition Pattern*. While this Pattern's identification and formalization is very recent, we have found many real-world systems (mostly human-made, not engineering systems, for the time being) that conform to the Pattern, and hence support the choice of calling it a "pattern" (or a "best practice"). Experience (see below) has shown that understanding and explaining the Pattern is easiest by refering to such human systems-of-systems (schools, cities, traffic, factories,...).

During Year 2 of the project, we have already been able to test the scientific value (and the pragmatic attractivity!), of the presented methodology (from an early stage on) on non-project researchers, developers and students, in the context of (i) two *European PhD Schools in Robotic Systems*[1] in September 2013 and January 2014, (ii) Master courses in *Embedded Control Systems* at KU Leuven and TU Eindhoven in Spring 2013, and (iii) dedicated and on-site one-day workshops at industrial development teams (Atlas Copco Belgium, and Tecnalia Donostia). This *"release early, release often"* approach has allowed to let the Pattern's description mature, but it has also shown that it can trigger drastic changes in the methodology of system developers, enabling them to bring a lot more structure in their system designs.
However, *the* major pragmatic lesson that we took home from these "outreach" activities, however, is that the consistent application of the Pattern is very difficult, even impossible, with the software architecture frameworks that are mainstream in the robotics community, that is, ROS and/or Orocos. In retrospect, this problem was easy to identify, because these frameworks do not support *hierarchical composition* of components, while this is an essential aspect of the Component Composition Pattern. This "lesson learned" has triggered the creation of a new framework, complementary to the mentioned ones, and that can fill the gap between a *system architecture model* and a *system architecture implementation*. This new framework, named `microblx`[2] is quite young and not yet mature, but it is already driving constrained-based motion applications on real robots. Its major added value is to allow extremely efficient realtime execution of functionalities, with instantaneous switches between different computational "schedules" and with the possibility to have work on all relevant data in a "shared memory" context, which is definitely a plus for highly integrated sensori-motor control.

In the generic context of cognitive robot systems, the consortium has focused on two principal directions of research (task execution and planning), and considered two principal embodiments of the cognitive capabilities (knowledge of the task, and sensing of the environment). More concretely, the following outcomes have been achieved: *i)* specification of compliant control in tasks space, *ii)* task-based specification of manipulation actions, *iii)* specification and execution of tasks where robots interacts kinaesthetically with an operator (co-manipulation), *iv)* task-based learning of manipulation actions, and *v)* sensor and model-based robust grasping synthesis. In fig. 1 is depicted how these specific achievements fit in the rough categorisation described above. The following Chapters of this Deliverable are conceived as follows:

- Chapter 1, *generic methodology of system architecture*: this Chapter provides motivated guidelines about how to architect knowledge-driven systems, capable of on-line adaptation and learning. The *Composite Component Pattern* is the new central concept in this methodology.

- Chapter 2, *specific approaches of "cognitive" system behaviour*: this Chapter summarizes and connects the partners' particular research results of project Year 2, in creating functionalities for such knowledge-driven motion planning, execution, adaptation and learning.

- Chapter 3, *collection of publications*: while the previous Chapters explain the "big picture", this Chapter collects all the publications that the Consortium produced, and that contain more detailed information.

[1] `http://phdschoolinrobotics.eu`
[2] `https://github.com/kmarkus/microblx`

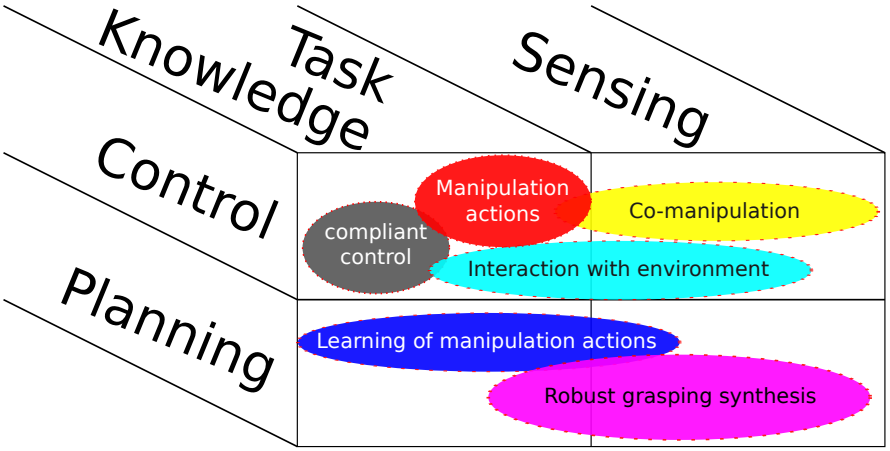

Figure 1: Research efforts charts: The research efforts of the consortium have focused on: *i)* the "behaviour" contexts depicted in the ellipses, and *ii)* the "structural" aspects of how to make robot architectures that can integrate all the aspects represented on the diagram's axes.

Chapter 1

System architecture methodology

In Year 2, KU Leuven's research has led to the concrete so-called *"Component Composition Pattern"* depicted in Fig. 1.3. The motivation behind this effort is that the mainstream practice of using "flat" system architectures (based on the simple ROS or Orocos "IO" components with publish-subscribe messaging in between, Fig. 1.1) leads to systems whose increasing number of functionalities, and especially whose increasing number of *interactions*, cannot be comprehended by one single system developer anymore. Hence, such system architectures become very "brittle": every time one adds new functionality, adds new data streams, taps existing data streams for new purposes, or re-deploys software components over a different computational and communication hardware, the *predictability* of the overall behaviour of the system decreases, and none of the system developers can still know for sure what the effects of the changes will be, or what new "glue code" will have to be created.

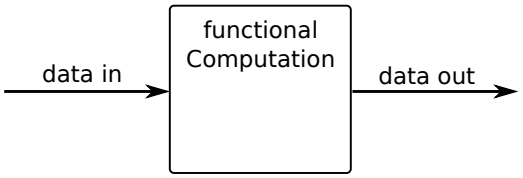

Figure 1.1: The simple "publish-subscribe" component primitive used in mainstream robotics system architectures. When trying to add "knowledge" to such components, developers are always (but most often not consciously!) confronted with the challenge of defining the *context* to which the components behaviour should be adapted, via application-dependent (hence, context-dependent!) learning and knowledge injection.

1.1 D3.2 in the context of the RoboHow system architecture

The RoboHow system consists of an abstract machine for plan execution plus a pipeline for knowledge acquisition and processing, driven by developments in UniHB. The architecture of the abstract machine that has been shown in the first review is depicted in Figure 1.2. UniHB are currently elaborating on this architecture in a journal paper draft that is in preparation; KU Leuven has started the work to find the best way to integrate the results in knowledge processing into the new Component Composition Pattern architecture, step by step.
As part of this architecture, WP3 investigates the design, implementation and evaluation of the

RoboHow motion control engine. In this section, we will propose a component architecture methodology that is targeted at the creation of the RoboHow motion control engine and system components at the implementation level needed for its operation.
At this stage of the research, it is not clear to what extent we will be able to realise the whole RoboHow system architecture in a reference implementation that will be "easy to use" for all interested parties, inside or outside of the project. *The* major hurdle in this context is the lack of mature *tool support* to help developers create new, application-specific, RoboHow system architecture that exploit the full potential of the integration of "reasoning", "learning" and "sensori-motor control" that the RoboHow project is producing.

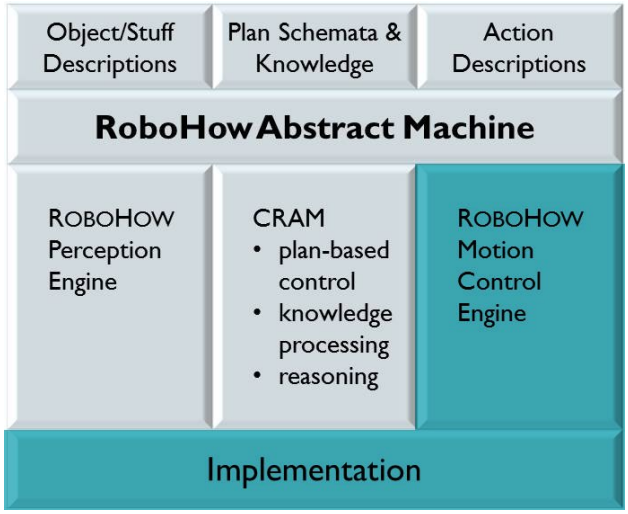

Figure 1.2: Components of the RoboHow abstract machine and contributions of WP3.

1.2 D3.2 in relationship to Work Package 7

Part of the work reported upon in this Deliverable deals with "system integration", and hence has strong interactions with Work Package 7 on *System Integration and Benchmarking*. And for example part of KU Leuven's person months efforts have been spent on supporting both Work Packages. The reporting about the system integration work is done in this Deliverable 3.2, because this Work Package 3 has a more *methodological* character, while Work Package 7's focus is on the *pragmatic* aspects of building prototype implementations of the RoboHow system architecture.

1.3 Composite Component software pattern

This Section presents a *composition software pattern* to help developers bring structure into their systems, and methodology into their design activities. More in particular, the role of "knowledge processing" in such architectures is highlighted.
The above-mentioned phenomenon of system developers loosing overall oversight and control in systems with growing behavioural and interaction complexity—especially when only having access to primitive system building blocks as depicted in Fig. 1.1—is not an exclusive property of computer-controlled systems, but a problem that human and animal societies have been coping

with forever already. The presented system composition software pattern is inspired by the solutions created in human-populated, knowledge-driven and highly interacting systems, such as factories, schools, traffic, countries, etc. In Year 2 of the project, we achieved to identify the full[1] Pattern, formalized it for use in the *structural* design of robot system architectures, and documented the *constructive* way of how to start applying the Pattern in new complex system designs.

A key factor in the architectural approach of human-built system is to introduce *hierarchy* and *context*, in "appropriate" ways. What can be considered "appropriate" is the outcome of centuries of trial-and-error (which is in many cases still ongoing, especially every time disruptive technologies are being introduced in society).

The system composition pattern of Fig. 1.3 attempts to formalize and document this hierarchy-and-context strategy, in order to introduce it into robotic system design.[2]

The first thing that Fig. 1.3 shows is that the presented pattern can be used in a *fractal, hierarchical* way: every component in the composition pattern can, in itself and in general, be considered as a "composite Component" that can be "exploded" into a new sub-system built on the basis of the exact same composition pattern.

Secondly, the pattern has a *structure* of component *activities* and (data) *interactions*, whose *role* we will explain by means of the example of a school in the human world. (Recall that this Chapter deals only with the *structural* aspects of system development; the *behaviour* of the system is realised by "plugging in" concrete algorithms into the activities and interactions: control, planning, perception, learning, etc.) The school example is chosen because it is *i)* sufficiently complex and "cognitive" to be worthwhile for the goals of the project, and *ii)* everyone is familiar with the example's "domain knowledge", to understand the motivations why the different parts of the composition pattern are needed, and how they have to interact. Here is the summary explanation of the pattern:

- *Functional Computations*: these are the algorithms that provide the "behaviour" needed by the system in order to realise its added value. *All* the other parts in the composite Component are "overhead", only necessary for realising these functionalities, in a structured way, with clear indications of roles and responsibilities.

  In the School example, the functional Computations are, in the simplest case, the learning activities of the pupils, the teaching activities of the teachers, the logistic activities of the non-educational staff.

- *Coordinator*: this is the *singleton* in the composite Component that is making decisions about when and why the system's behaviour has to change.

  In the School example, the Director of the school fulfils this role: (s)he decides whether and when new staff has to be hired, where they have to fit in, with whom they have to cooperate, and what activity they should deploy. Note that the Director does *not* indicate *how* these required behaviours should be realised, since that would introduce too much "coupling" between the Director's own behaviour and that of the other staff in the school.

[1] The project's focus on representing knowledge, and integrating it into robot control architecture, has taught us enough modesty to be always ready to apply the "open world assumption" to our own research: if new use cases turn up that bring arguments to improve the Pattern, we have no legacy investments or mental inertia not to do so!

[2] We have no reasons to believe that the pattern would not hold equally well in the more generic context of so-called *"cyber-physical systems"*.

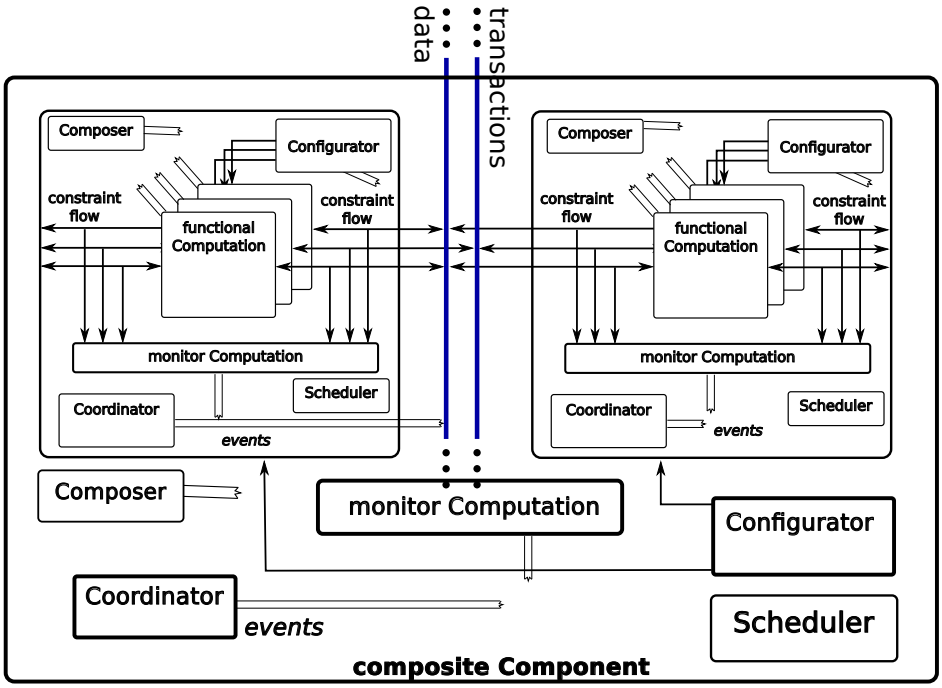

Figure 1.3: This *software architecture pattern* is introduced to help system developers, in a motivated and methodological way, to make the *trade-offs* that are *always* needed when putting a complex system together from a large set of available hardware and software components. *The* major trade-off to make is how to distribute the various *functional computations* that the system requires over communicating software components, while still having control over the overall, application-directed coordination of all these components and the data exchanges that they require.

- *Configurator*: this is a component that is responsible for bringing a *set* of functional Computational components into a configuration that makes them—*together* and in a synchronised way—realise a different system behaviour. There can be multiple Configurators, each dealing with a disjoint subset of functional Computational components. Configurators are triggered by the Coordinator, and they shield the latter from having to know *how* the behavioural change that is expects is realised exactly by the available Computational components.

  In the School example, Configuration takes place by the staff person responsible for assigning teachers to classes and lecture schedules, and operational staff to concrete logistic roles.

- *Composer*: this is the software activity that creates a new *structural* model of the composite Component, that is, it introduces new data exchanges and functional components, couples existing or new functional components to existing or new data flows, etc.

  In the School example, this activity takes places when the school building is being built or renovated, when new personnel is hired, existing personnel is fired, or new technologies are introduced that are visible to pupils, teachers and/or staff.

- *Scheduler*: this is the activity that *triggers* the functional Computational components when they are needed to realise the system's overall behaviour. Like the Coordinator, there should be only one Scheduler active within a composite Component.

In the School example, this can be realised by the school's bell system coupled to the computer that runs the algorithm that computes (or just stores) the schedules of all lectures of each day.

- *Monitor*: these are the software activities that play a dual role with respect to the Configurators: while a Configurator takes action such that the system's behaviour can *expected* to be what is required, the former checks whether the *actual* behaviour corresponds to the expected one. As with Configurators, one single Monitor can be responsible for a *set* of functional Computational components. Whenever the discrepancy between expected and actual behaviour becomes "too large" (which should be a *configurable* threshold!), the Monitor fires an event, to which all other components can react.

  In the School example, this role is taken up by the teachers, by means of written or oral tests and discussions.

- *Data* exchange: while "publish-subscribe" is only one particular, uni-directional, way of exchanging information between components, one should in general allow bi-directional alternatives, with other exchange policies than just publish-subscribe; for example, data buses, shared memory, ring buffers, lock-free buffers, etc.

  The data exchange should not be limited to only the *functional* data, but components should also keep each other informed about *operational* data: how well is each component performing the functionality that the connected components expect it to perform? how close to its resource limits is this component operating? how much progress has already been made towards set targets for each component? Etcetera.

  In the School example, one sees the functional data exchanges taking place via communication infrastructure in the form of: post boxes for the teachers and the administrators; announcement boards; audio communication via loudspeakers in each class; fire alarm; etc. The operating data is exchanged in regular job satisfaction interviews, or teacher meetings, etc.

- *Transactions*: some data has to be archived to so-called *persistent storage*, from time to time, in order to allow the later inspection of the system's activities.

  In the School example, these are the intermediate and final reports of the pupils, as well as the financial records of the school administration.

All of the above activities and interactions are "deployed" into one *explicitly* indicated *context*, that of the "composite Component" (which should be chosen as a "stable" sub-system in the overall system). "School" was the context of the explanations above, but it is easy to transpose these explanations to other well-known contexts of complex human activity interactions, such as factories, traffic, or cities.
These examples all show the relevance of "hierarchies" of "stable sub-systems": a factory consists of many production lines, each having several production work cells; traffic has cars, traffic lights, traffic jam observers and information displays; each of those is a full system in itself, and they are "stable" in the sense that they can be deployed in almost all possible factory or traffic situations, everywhere in the world, and for every production or transportation goals. The robotics domain has not yet reached this phase of having such stable sub-systems readily available, or even explicitly identified or agreed upon; this Work Package's ambition within the project is exactly to realise

a *step change* in how we compose functionalities into components, such that the mentioned requirements are satisfied with a higher robustness, performance and programmability.
The role of the composite Component's context can not be overestimated: it indicates a *boundary* (or so-called "closed world") in which particular knowledge can be applied, or particular learning can be integrated, *without* having *i)* to integrate the knowledge inside *one particular* component within the composite, and *ii)* to care about taking into account "reasoning rules" that are not relevant to the context. When an explicit context primitive is lacking in system architecture design, these "closure" and "introspection" aspects of a sub-system always end up somewhere inside one or the other component, that then has too much knowledge about the inner workings of the other components to be practical for later scaling and/or further integration of this particular sub-system.

1.4 The "Hierarchical Hypergraphs" meta model

As already mentioned a couple of times, this Work Package has spent a good amount of its research efforts on the *formal* description of complex system architectures. This drive toward formalization resulted in a (draft) publication, **??**, that introduces a *meta model* to represent the purely structural aspects of system architectures, with very broad scope: software systems, Bayesian networks and other probabilistic models, control diagrams, finite state machines, etc. It also allows to model *knowledge contexts*, which is very relevant in this part of the RoboHow work; Fig. 1.4 depicts how different contexts can overlap the same "lower level" sub-system.

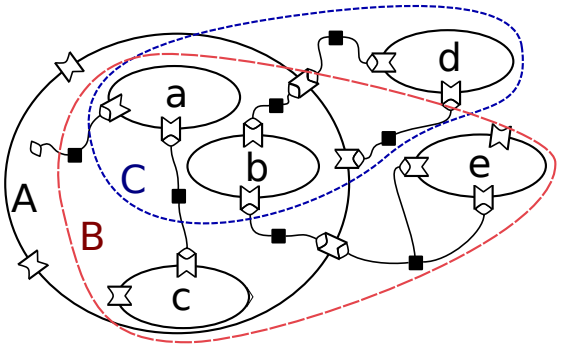

Figure 1.4: An example of a hierarchical composition in which *containment* does not follow a strict *tree* hierarchy: the compositions with the small dashes (blue) and the long dashes (red) have some internal nodes in common. This use case fits very well to modelling overlapping contexts of knowledge, applied to the same sub-system.

1.5 Integration of knowledge and learning

This Section explains how knowledge and learning can be integrated into the software of a complex robot system. The core idea behind the methodology is that "behaviour follows structure": a systematic application of the structural system composition pattern of Sec. 1.3 makes it a lot easier to select the sub-set of components, interactions and data that a reasoning system is responsible for adding knowledge to. Or, similarly, to give the set of parameters and models that a learning algorithm will have to work with (as "prior inputs" as well as "output" of its learning).

Figure 1.5 shows the "bird's eye view" on a knowledge-driven robot application, that is, it brings a "meta level structure" of four different aspects that have to be integrated: knowledge, algorithms, system architecture, and hardware/software infrastructure.

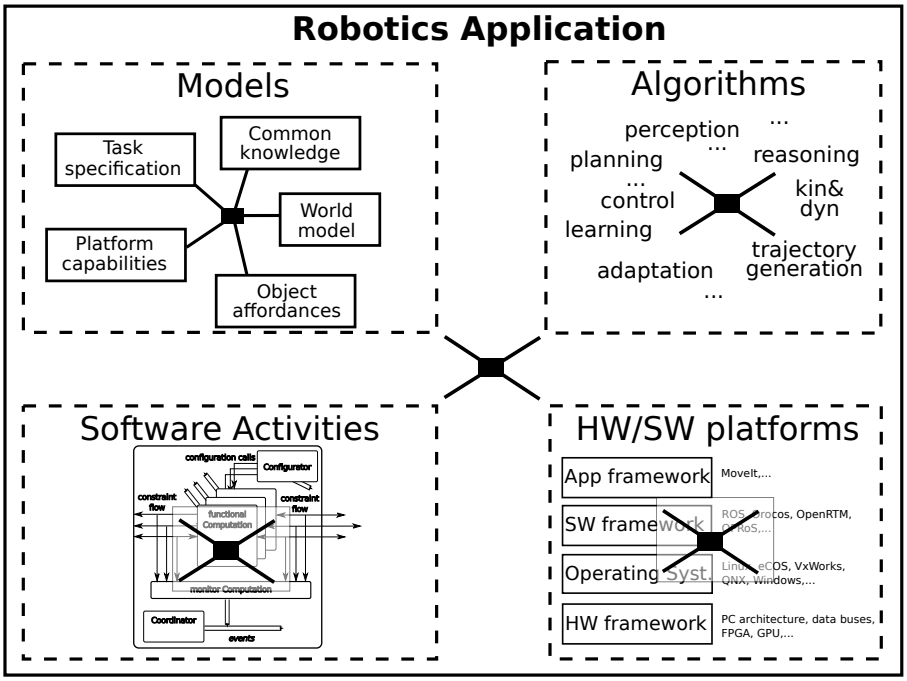

Figure 1.5: The various aspects that a system architecture of a cognitive robotics application must be able to integrate: the top-left corner lists the various complementary sources of *knowledge* (in the form of formal "models"); the top-right corner contains all computational algorithms available to the system developer; the bottom-left corner provides knowledge about which *software architectures* are available at the *application* level; and the bottom-right corner provides descriptions of the hardware and software *infrastructure* level. The black nodes with arrows represent the *integration* that must take place between these several aspects, at various *levels of abstraction* of the knowledge, algorithms and data structures involved in the integration.

With our current understanding of the problem of making such complex systems, it is not (yet. . . ?) possible to provide more explicit *guidelines* about *how* exactly one specific piece of knowledge, or one specific learning algorithm, should be integrated. In addition, we adhere to the mainstream "paradigm" that *"My architecture is always better than your architecture"*, that is, it is impossible to make generic statements about how exactly *the* system architecture of a particular application should look like.
Anyway, the *generic guideline* to integrate knowledge into a system, built with the Composite Component Pattern from the ground up, is the following:

- the structure of the Pattern identifies clearly the roles and responsibilities of sub-components: Coordinator, Composer, Scheduler, etc. This allows to focus the knowledge integration to small sub-parts of the system.
- each of the mentioned sub-parts can be connected to a knowledge server in two ways:
  - it gets the functionality to "query" a relevant knowledge base when necessary;

  - a knowledge base that has access to the explicit and formal representation of the structural connections in a system has an easier job in "injecting" its knowledge in the right places. The latter are typically the Coordinator and Configurators.

- the explicitly modelled *composition boundaries* allow for focusing of the knowledge bases that are relevant in particular contexts.

  In addition, the HH-NPC to model allows to have multiple contexts overlapping each other, which is a necessity for the "open world" driver in knowledge-based systems.

The integration of learning into a system architecture follows the "opposite way" of that of knowledge: the running system updates the knowledge base with newly acquired information. From the system architecture point of view, this updating process can be supported by the same primitives as for knowledge integration: context-aware "querying" or "configuring" from small-scale sub-components in the system.
None of this potential has already been explored in real reference implementations; this exploration is scheduled for Year 3.

1.6 Integration of "legacy" components

One of the outcomes of the presented research on system architecture design is that the sub-system composition pattern of Sec. 1.3 is not only a very useful *design* aid, but that it can also help *to refactor* existing "legacy" code towards Composite Component Pattern compliance, *one step at a time*. Indeed, this is a positive side-effect of the Pattern's core focus on "peer-to-peer" integration, on the basis of the above-mentioned limited set of "operational data" exchange.
The strategy within the project is to let KU Leuven work with the other partners in realising such incremental architectural improvement, refactoring and integration process, based on the composition pattern, and driven by the explicit knowledge and learning contexts of the sub-systems.
Figure 1.6 shows a first example of this strategy, in which we are (re)building the *constrained-based motion control stack* for our robots, in cooperation with CNRS and Bremen; more details of this activity are given in Deliverable D.3.1 (*"Documentation about the constraint-based framework"*.
As a more detailed example of how the Component Composition Pattern can be used on real systems, Figure 1.7 presents a detailed view on the innermost composite component of a KUKA youBot motion control implementation.

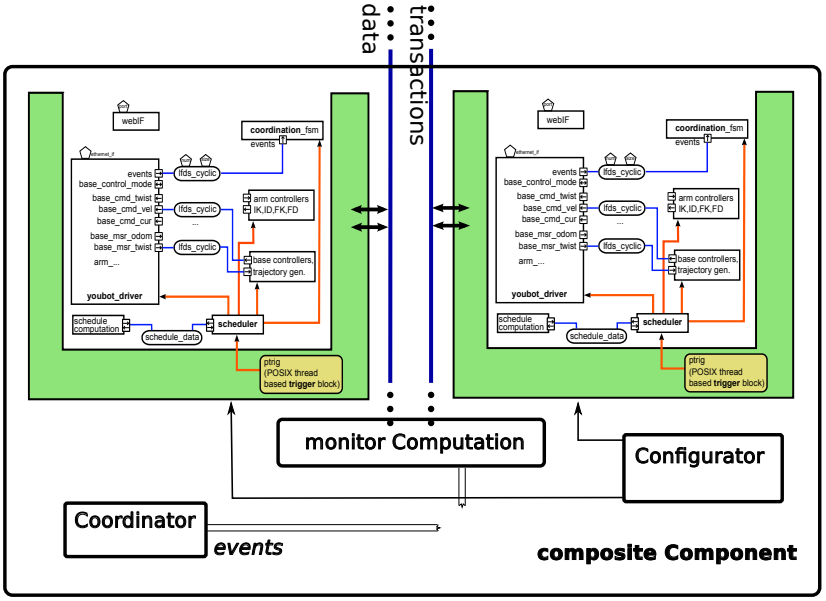

Figure 1.6: This Figure shows an example of the generic pattern of Fig. 1.3, applied to the (still rather simple and cognition-less) motion control of *two* robots whose motions are coordinated. The blocks inside the functional components represent the data interactions between a particular motion control algorithm, and the drivers needed to access the robot's hardware and inter-process communication channels.

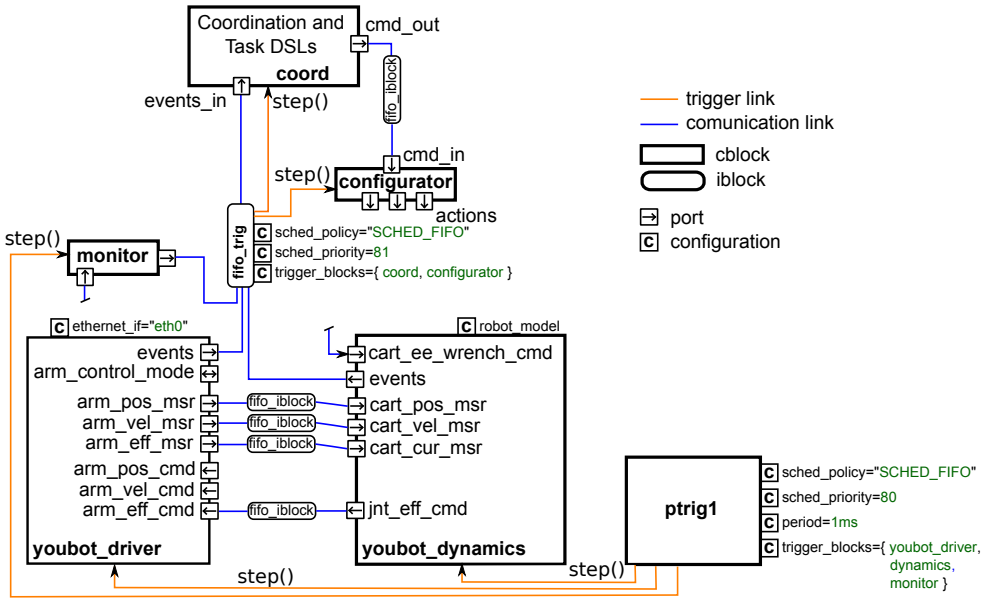

Figure 1.7: This Figure shows a concrete example of a computational composition for the realtime control of, in this case, a KUKA youBot. This composition is implemented with the `microblx` framework; note the explicit "scheduling", via `step()` triggers, in different parts of the architecture, and driven by different causal sources.

Chapter 2

Overview Year 2 R&D results: Arm and hand manipulation

All the tasks which involve interaction with the environment make use of a manipulation action, at some point: for this reason, the research of robust and flexible approaches to describe and execute such tasks is of paramount importance within the project, and is the intersecting point of different research lines, that tackle this problem from complementary points of view.
The manipulation tasks that are investigated in the RoboHow project are executed in only partially known environment, on objects whose shape is irregular and whose position can vary; or, in the case of co-manipulation tasks [1, 11], the robot interacts with a user whose intentions are transmitted to the robot as visual or haptic cues.
For all these reasons, both the task specification (achieved either by reasoning or by teaching) and the task execution must incorporate some degree of cognitive capabilities, in such a way the system is able to cope with ever-changing tasks, reacting to unexpected situations.
This Chapter introduces the Consortium's progress in providing improved "behaviour" (control, planning, reasoning, learning); the integration of the various contributions into more coherent, performant and predictable (sub-)systems is still work in progress. In order to structure the progress reporting in self-contained sections, we will classify the work in two categories: Control (Sec. 2.1) and Planning (Sec. 2.2).

2.1 Hand and Arm Control

Executing a motion consists, in our view, in fulfilling a constraints, optionally minimizing a cost function. This general paradigm allows for flexible control design: by expressing constraints and cost function on different variables (such as forces and velocities, expressed in Cartesian, joint, or other spaces) and optionally modifying the relative weights, gains and other parameters, it is possible to obtain very complex behaviours.
Constraint-based frameworks incorporate sensor-based and reactive control in a natural fashion: by considering, in the constraints definition, geometric (or kinaesthetic) features extrapolated by data collected by sensors (e.g. forces, camera- or range finder- captured positions, camera plane informations, *etc*), it is possible to design reactive behaviours that allows the robotic systems to automatically adapt to unmodelled, uncertain or changing characteristics of the environment.
The simpler example of compliance to an unknown environment are force and impedance modes: for this reason, in [4] force and impedance controls have been reformulated within the iTaSC

framework. More examples include traditional obstacle avoidance, but also active camera head tracking, [11] and co-manipulation with visual tracking, [1].
At this level, there is no or very little "memory" of previous states (since the adopted control is based on the instantaneous optimization paradigm), and long term execution is decided in a planning stage, which interfaces to this level by means of task descriptions that revolve around the task space[1]. In this way we are able to divide the responsibility of the control algorithms (that acts in in a greedy way, pursuing local objectives) from the planning algorithms (that, on the contrary, can works on whole actions).
Generally speaking, tasks are not described in terms of the robot (e.g. joints positions), but in terms of the task geometry (e.g. object frames). In case of manipulation and grasping tasks, it is convenient to expand the set of primitives employed in task definition, in order to easily describe the hand/grasp and arm configurations, e.g. the synergy that represents hand postures, or the manipulability space of the arm, as shown in [4].

2.2 Planning

The plan that is executed by the control algorithms is synthesized on a higher level, either employing a reasoning engine (*e.g.* the CRAM system), or a by learning, thus taking the role of the Coordinator element depicted in **??**. While this architecture is applicable to general tasks, once the focus is shifted toward manipulation tasks, new aspects (somehow connected to the interaction with objects) must be accounted for: for example, in [2] is shown how to build a probabilistic model of grasp stability, that allows to foresee which is the best "strategy" to execute a given task. The model is build with several steps that include simulation (in order to produce good a set of initial grasps), human "tutoring", and robot trials that are evaluated in function of visual and kinaesthetic stimuli. Instead, in [**?**] a more simulation-based approach is pursued; this approach greatly simplifying the learning stage, at the cost of relying only on 3D object models.
Also forces exerted by the robot play a central role: in [8, 9] is shown how positions and forces are equally relevant for learning and execution of simple cooking-related tasks. Again, the spaces that are used as references are task spaces. As such, forces and positions are expressed in object frames, while the robot kinematics does not play any role: in this way the learned force and position trajectories are independent by the robot, and robust to changes in the scenes.

Another example where cognitive aspects assumes a central role in the manipulation is the interaction with objects whose kinematics are largely constrained, for example, when opening a drawer or a door, [6]. In this case, the robot (instead of learning from a demonstration) "learns" the constraints offered by the environment, and use it in planning, or in reactive control, or both.
All these works share a common aspect that naturally emerges from manipulation tasks implementation: the need to find a space that simplifies the expression or the execution of the task, either *i)* by reducing the number of parameters needed to describe it, *ii)* or by looking to the correct space to represent it. While the above criteria are of general application in task specification, in the case of manipulation, we have more aspects to take into account, arising by the intrinsic complexity of robotic hands, grasp definition, and the interaction with the environment.

[1] see also Deliverable D3.1.

Chapter 3

Forewords to attached publications

This Deliverable summarizes the works that project consortium members carried over in the first two years of the project. As most of the works is already documented in the form of papers and articles, we will simply highlight how the achieved results concur toward the synthesis and execution of grasping manoeuvres. Note how these works describes algorithms that are can (and in many cases, are) encapsulated in the components depicted in fig. 1.3 and described in chapter 1: the papers practically describes either how to **compute**, how to **configure**, or how to **monitor** one or more aspect in which manipulation or environment interaction are involved. The contributions will be presented following the rationale presented in the previous Chapter, divided in Control and Planning.

3.1 Control and constraint specification

This Section briefly revises the works that are related to control by constraints in manipulation. We consider *i)* specification of force and impedance control as constraints, *ii)* specification of arm and hand poses, *iii)* co-manipulation with a human operator, and *iv)* compliant interaction with kinematically constrained objects.
In the first two cases we focus on the description of methodologies and extension to constrained-based frameworks toward manipulation-centred tasks, while the latter two focus on to blend of force, vision and a priory knowledge to comply with human desires and with the environment.

Structural system architecture

Reference work:

- Herman Bruyninckx, Markus Klotzbücher, and Hugo Garcia. Hierarchical hypergraphs of nodes, ports and connectors (hh-npc): a composable structural meta model for robotics. *Journal of Software Engineering in Robotics*, 2014. Draft

Force and impedance control

Reference works:

- Gianni Borghesan and Joris De Schutter. Constraint-based specification of hybrid position-impedance-force tasks. In *IEEE Proc. of the Int. Conf. on Robotics and Automation*, 2014. Accepted

One of the simplest and most used way to move in an unknown environment, is to react to kinaesthetic stimuli. Impedance control and force control allows to do so efficiently, ruling interactions either in function of displacement of a compliance system, or in function of forces exerted by/to the environment. In this work, we bring the structure of the task within the impedance control framework, by extending the capability of tasks frame formalism to specify desired force along Cartesian directions to more general definitions of output space.

Constraint-based specification of arm and position control

Reference work:

- Gianni Borghesan, Erwin Aertbeliën, and Joris De Schutter. Constraint- and synergy-based specification of manipulation tasks. In *IEEE Proc. of the Int. Conf. on Robotics and Automation*, 2014. Accepted

In this work we bring within the constraint-based approach the concept of synergies. In general, synergies are a method to reduce the dimensionality of a data set in which some correlation between the data exists. In the case of grasping, synergies allow to represent a grasp along some directions that could have also a semantic meaning. In task specification, this is a very useful characteristic, since it allows to abstract from the joint space representation, thus *i)* reducing the knowledge needed to program the task, and *ii)* separating the declaration of the desired grasp (that can in principle be achieved on hands that differ in kinematics) from the implementation.
Following the same rationale, we also consider how to control the stance of the arm, in order, for example, to maximize the manipulability in one preferred direction in such a way it is easier to execute a task.
Reference work:

- Azamat Shakhimardanov and Herman Bruyninckx. Vereshchagin's algorithm for the linear-time hybrid dynamics and control with weighted or prioritized partial motion constraints in tree-structured kinematic chains . In *IEEE Conf. on Intel. Robots and Systems*, 2014. Submitted

This work introduces a new integrated solver for acceleration constrained motion tasks for tree-based robots, that can deal with priorities and weighting between constraints in the same linear-time sweeps over the robot's kinematic chain. Its reference implementation is designed according to the *Component Composition Pattern*.

Human-Robot Co-manipulation

Reference works:

- Dominick Vanthienen, Tinne De Laet, Wilm Decré, Herman Bruyninckx, and Joris De Schutter. Force-Sensorless and Bimanual Human-Robot Comanipulation Implementation using iTaSC. In *Proc. of 10th IFAC Symposium on Robot Control*, pages 832–839, 2012

- Don Joven Agravante, Andrea Cherubini, Antoine Bussy, and Abderrahmane Kheddar. Human-Humanoid Joint Haptic Table Carrying Task with Height Stabilization using Vision. In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2014

- Yiannis Karayiannidis, Christian Smith, Francisco Vina, and Danica Kragic. Online kinematics estimation for active human-robot manipulation of jointly held objects. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4872–4878, 2013

In tasks where the robot cooperates with an operator, sensing plays a central role. In the brought examples, we consider three different approaches to solve the problem of a person and of a robot jointly executing the task of moving a plank: in the first example, the robot is driven by kinaesthetic stimuli. In the second, visual cues are employed to estimated the actions that the user applies to the plank. In the last work, the emphasis is posed on estimating a very simplified kinematic model of the operator, in such a way it is possible to better react and plan the task in such a way the user "constraints" are not violated.
These examples expose the complexity in defining seemingly trivial tasks: first of all, in order to successfully execute the principal task, many other constraints must be enforced, like to maintain self balancing or avoid obstacles. Secondly, the robotic system must be equipped with cognitive capabilities (intended as sensors and algorithms that manipulates the acquired raw data) in order to interpret the environment (*e.g.* to estimate poses of objects of interest) and the intention of the operator, and in order to react in function of the acquired data.

Interaction with kinematically constrained objects

Reference works:

- Y. Karayiannidis, C. Smith, F.E. Viña, P. Ogren, and D. Kragic. Open sesame! adaptive force/velocity control for opening unknown doors. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4040–4047, 2012

Along with the interaction with operators, robots can interact with objects that are somehow constrained, e.g. performing door opening operation. In these cases, by employing a simple general model and estimating online the parameters of the specific object, it is possible to reduce interaction forces.

3.2 Cognitive planning in the task space

In this section we will give an overview of works that deal with planning and learning. The most noteworthy characteristic that is shared by these works is that the task description becomes more and more central, and some *a priori* knowledge is used to describe the actions composing the tasks or the learning algorithms that observe task executions.

Learning (Tasks) by Demonstration

Reference works:

- A. L. Pais, Keisuke Umezawa, Yoshihiko Nakamura, and A. Billard. Learning robot skills through motion segmentation and constraints extraction. HRI Workshop on Collaborative Manipulation, 2013

- A. L. Pais and A. Billard. Extracting task constraints as a middle layer between low level control and high level planning. RSS Workshop on Programming with constraints, 2013

These two works address the problem of teaching by demonstration of tasks executed in kitchen environment where forces exerted by means of the robot with a tool play an important role.
To understand the centrality of the task, we can observe that in this works the learning algorithm knows which are the object/frames of interest for the whole task sequence, and employs a space (in which forces and poses are represented) for each of such objects: by comparing the same motion and force profiles, captured during human demonstrations, and represented in each of these spaces, it is possible to create a description that capture the meaning of the task, and thus allows for a reproduction of the task, even in a mutated or different environment. On the contrary, the same techniques applied in a space that is not related to the task, (e.g. joint space trajectories) fail to show the robustness achieved in these conditions.

Grasp planning

Reference works:

- Yasemin Bekiroglu, Dan Song, Lu Wang, and Danica Kragic. A probabilistic framework for task-oriented grasp stability assessment. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3040–3047, 2013

-

The last topic that we covered is the grasp synthesis. The addressed question is how to position the wrist and the fingers is a way it is possible to move an object without losing it. The problem is addressed from two different perspectives: in one case the grasp strategies are trained in different steps: simulation, try-and-error, and tutoring, and then the trained model are used to execute the task, aiming a stable grasp. On the contrary, in the second work, grasp stability is assessed by simulation only, but grasp can be optimized also toward other factors (e.g. minimize finger joint torques), while in the first case the only maximized factor is the probability to achieve a stable grasp.
In both cases, the systems have *a priori* knowledge of the grasp strategy that is valid for a class of objects, that, once the acquisition/learning process is concluded, can be applied in real scenarios.
Open Access versions of the following publications are accessible via `http://www.robohow.org/publications`.

Cited Works

Open Access versions are accessible via *www.robohow.org/publications*

[1] Don Joven Agravante, Andrea Cherubini, Antoine Bussy, and Abderrahmane Kheddar. Human-Humanoid Joint Haptic Table Carrying Task with Height Stabilization using Vision. In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2014.

[2] Yasemin Bekiroglu, Dan Song, Lu Wang, and Danica Kragic. A probabilistic framework for task-oriented grasp stability assessment. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3040–3047, 2013.

[3] Gianni Borghesan, Erwin Aertbeliën, and Joris De Schutter. Constraint- and synergy-based specification of manipulation tasks. In *IEEE Proc. of the Int. Conf. on Robotics and Automation*, 2014. Accepted.

[4] Gianni Borghesan and Joris De Schutter. Constraint-based specification of hybrid position-impedance-force tasks. In *IEEE Proc. of the Int. Conf. on Robotics and Automation*, 2014. Accepted.

[5] Herman Bruyninckx, Markus Klotzbücher, and Hugo Garcia. Hierarchical hypergraphs of nodes, ports and connectors (hh-npc): a composable structural meta model for robotics. *Journal of Software Engineering in Robotics*, 2014. Draft.

[6] Y. Karayiannidis, C. Smith, F.E. Viña, P. Ogren, and D. Kragic. Open sesame! adaptive force/velocity control for opening unknown doors. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4040–4047, 2012.

[7] Yiannis Karayiannidis, Christian Smith, Francisco Vina, and Danica Kragic. Online kinematics estimation for active human-robot manipulation of jointly held objects. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4872–4878, 2013.

[8] A. L. Pais and A. Billard. Extracting task constraints as a middle layer between low level control and high level planning. RSS Workshop on Programming with constraints, 2013.

[9] A. L. Pais, Keisuke Umezawa, Yoshihiko Nakamura, and A. Billard. Learning robot skills through motion segmentation and constraints extraction. HRI Workshop on Collaborative Manipulation, 2013.

[10] Azamat Shakhimardanov and Herman Bruyninckx. Vereshchagin's algorithm for the linear-time hybrid dynamics and control with weighted or prioritized partial motion constraints

in tree-structured kinematic chains . In *IEEE Conf. on Intel. Robots and Systems*, 2014. Submitted.

[11] Dominick Vanthienen, Tinne De Laet, Wilm Decré, Herman Bruyninckx, and Joris De Schutter. Force-Sensorless and Bimanual Human-Robot Comanipulation Implementation using iTaSC. In *Proc. of 10th IFAC Symposium on Robot Control*, pages 832–839, 2012.