TOWARDS DESCRIBING AND DEPLOYING WHOLE-BODY GENERIC MANIPULATION BEHAVIOURS

José de Gea Fernández[1], Dennis Mronga[1], Malte Wirkus[1], Vinzenz Bargsten[2], Behnam Asadi[2] and Frank Kirchner[1,2]

[1]*DFKI, Robotics Innovation Center, Bremen, Germany*
[2]*University of Bremen, Robotics Group, Bremen, Germany*

Key words:

artificial intelligence, AI, manipulation, skill, software architecture, task planning, monitoring, sensor-based control, dynamic control, whole-body control, planning

ABSTRACT

This paper presents our current work on integrating three key components aiming at significantly increasing manipulation performance: namely, trajectory planning, whole-body sensor-based reactive control, and dynamic control into a newly developed modular, robot-independent and easily-reconfigurable software framework which allows reusing components to describe a variety of complex manipulation behaviours.

INTRODUCTION

A historical overview of the dexterous robotic manipulation problem will promptly show that the required control components had been long ago identified: high-level task planning for defining the subtasks to perform; contact, phases, and events detection basically by incorporating multiple sensor modalities, and low-level dynamic controllers which execute the required actions [1]. Recent developments in dexterous robotic systems ([2], [3], [4], [5]) show impressive capabilities, mostly by excelling at one or two of the previously mentioned key components. However, we believe that the next significant leap in dexterous manipulation will emerge by endowing a complex robot with a flexible software framework which coherently integrates all those individual crucial components. Moreover, the software framework needs to allow reusing components to create new behaviours and to transfer those behaviours to robots of different morphology or hardware.

The project BesMan ("Behaviors for Mobile Manipulation") [6] is a joint project of the the Robotics Innovation Center of the German Research Center for Artificial Intelligence (DFKI) and the Robotics Research Group of the University of Bremen (UoB) funded by the German Space Agency (DLR) with federal funds of the German Federal Ministry of Economics and Technology. The goal of the project is the development of generic dexterous manipulation strategies which are independent of a specific robot morphology and suited both for one and two-arm robotics systems. Furthermore, novel, situation-specific behavior shall be learned by means of a learning platform [7]. The development of dexterous manipulation procedures is mainly the responsibility of the DFKI while the UoB develops the learning platform. In this paper, we focus mainly on reporting the status of first part: the manipulation control strategies.

SOFTWARE ARCHITECTURE

As shown in neuroscience studies, human manipulative skills are not the result of high powerful actuation systems or fast sensorimotor loops. On the contrary, the key lies on the task organization, involving sequences of organized action plans at various levels of complexity [8]. In that sense, manipulation is considered a succession of phases bound by events which simultaneously act as "borders" between those phases or subtasks. Thus, we can see dexterous manipulation as an event-based action sequence. In essence, the brain forms action plans in terms of a sequence of subtasks but it is the sensory system the one in charge of certifying that the expected sensory events are occurring, that is, that the subtask goals are being achieved.

Taking the previous ideas into practice, we see that event-based task planning is one of the key features of task control in humans. Similarly, the recognition of those events is of importance to create the transitions between manipulation phases as well as serving as monitoring / supervision of the task at hand. In this area, DFKI have been developing the software toolchain ROCK [9], which is a software framework for the development of robotic systems. The underlying component model is based on the Orocos RTT (Real Time Toolkit). ROCK provides all the tools required to set up and run high-performance and reliable robotic systems for wide variety of applications. It includes the plan manager Roby, which is an event-based plan management tool offering a seamless integration of planning and plan execution activities. Basically, Roby integrates plan generation, plan analysis, and plan adaptation within a coherent framework [10].

Within the project BesMan, the (minimum) required components for such a generic robot software framework for robotic manipulation have been identified and are shown in Figure 1. The Figure highlights schematically the main software modules running inside the robot, which additionally has connections to both the 'Learning Platform' (mostly offline, running outside the robot) [7] and the ‘Action Plan' in order to receive a sequence of subtasks. Both components are outside the scope of this paper as their information is assumed to be received by the robot externally. However, we will see in the next section how to read and parse the information coming out of the action plan to create a time sequence of robot actions which generates the desired complex manipulation behaviour.

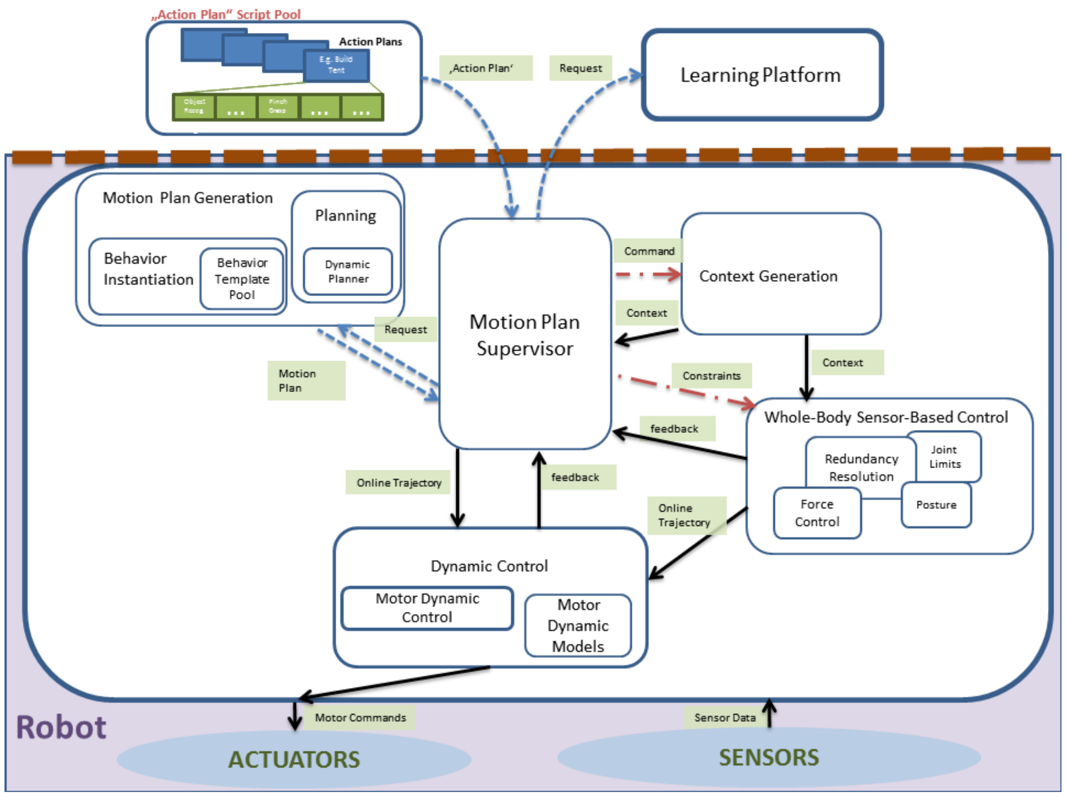


Figure 1. Main components of the robot software architecture

In essence, the role of the main components is:

- *Motion Planning.* This component is responsible for trajectory generation in the Operational Space. It will take into account kinematic constrains in a first phase, which later on will be extended to included dynamic constraints. Moreover, this component must take into account obstacles and be able to respond in real-time to changes on the environment.

- *Dynamic Control.* This component is in charge of the dynamic control. It uses robot dynamic models to control the robot and achieve the task.

- *Multi-sensor whole-body reactive-based control.* This component is in charge of the whole-body reactive control. It uses the available degrees of freedom of the robot to regulate the main task, posture, and external forces/torques in order to comply with the main task and its associated constraints (joint limits, object orientation constraints, postures, etc …) and to optimally resolve the redundancy problem at run-time. In essence, this component deals with sensor-driven corrections and control of ongoing tasks. On the one hand, the component can be used to allow sensor-driven behaviours as visual servoing (use of camera information to directly drive the arm). On the other hand, its major focus is on making use of all the degrees of freedom of the robot in the most optimal way.

- *Context Generation.* This component provides information about the context in which the robot is located and any information which can be used to recognize the location and situation of the objects which are relevant for the task to be fulfilled. However, this component is not on the focus of the architecture (and the project) and will be solved by available methods to provide the robot with all required context information.

- *Behaviour Template Pool.* A behavior can be regarded as any action of an organism that changes its relationship to its environment. On the other side, a skill is a learned ability of an organism or artificial system. Skills can only be learned by behavior, but a skill is not the learned behavior but a template for an adapted production of a feasible behavior for situations that are to some degree similar to those in which it was learned. For those reasons, the robot architecture stores behavior templates in a common pool, to be instantiated and re-used when required, according to the current context information.

Action Plan

As previously mentioned, the second message we learn from neuroscience studies is the concept of sequence of actions plans which are used to create more complex actions [8]. We embrace this concept in our project BesMan and define an ‚Action Plan' as a sequence of subtasks which need to be followed in order to accomplish a certain complex task (for instance, building a specific infrastructure on the Moon). Thus, the subtasks of the ‚Action Plan' can be mapped to behaviour templates available on the robotic system. The mapping between those subtasks and behaviour templates would be pre-defined. Those behaviour templates are learned with respect to certain tasks and in certain situations. However, templates are to some degree adaptive and will adapt to a certain situation when executed. The behaviour instantiation will map a certain behaviour template for a given context to the appropriate motion plan. Finally, those behaviour templates will be available in a so-called Behavior Pool for its use when required.

In order to fulfill such a generality, a software framework has been developed to describe and control robot manipulation behaviors [11]. To keep independence from particular robot hardware and an explicit statement of areas of application, an embedded domain specific language (eDSL) in the host language Ruby is used to describe the particular robot and a controller network that drives the robot. Thus, we make use of a) a component-based software framework, b) model-based algorithms for motion- and sensor processing representations, c) an abstract model of the control system, and d) a plan management software to describe a sequence of software component networks that generate the desired robot behavior. Using component-based software architectures, the robot's behavior is created by the interconnection of the individual running software components; the resulting behavior is defined by the component network's topology and the configuration of the components. On the one side, re-using software components can save a lot of time, but of special interest is the ability to describe manipulation behavior only once and being able to transfer this behaviour between robots of different morphology and/or hardware.

The process of deploying the proposed framework is outlined in Fig. 2. It is divided into three steps. In the first step (Model), a set of models is created, basically algorithm-specific models for motion representation and object detection. In the second step (Combine), and by using the eDSL language, these elements are automatically related to each other to build compounds of motion models, virtual control systems, sensor processing models and sensors. These compounds implicitly describe the software component network that generates the desired robot behavior. On the base of these two steps, a broad range of different system behaviors can be described. For a practical application, coordination of these different behaviors over a longer time scale is required. This coordination is described in the third step called Concatenate. Here component network descriptions are arranged chronologically.

As previously mentioned, the development is supported by an embedded domain specific language (eDSL). The available commands of this language are used to describe a multi-level controller network. At each level, subsystems of the robot are defined and added to a virtual control system. A robotic system is herewith treated abstractly as a composition of several subsystems (multiple arms, head, torso, mobile base, etc …). In order to allow the transfer from robot behavior descriptions from one robot to another, the data processing chain is further divided into three independent and isolate areas: a sensor processing pipeline extracting contextual information from a sensor, a motion plan which describes the motion to achieve a certain task, and the robot control commands coming from a network of control components and which will translated into specific commands for the specific robot.

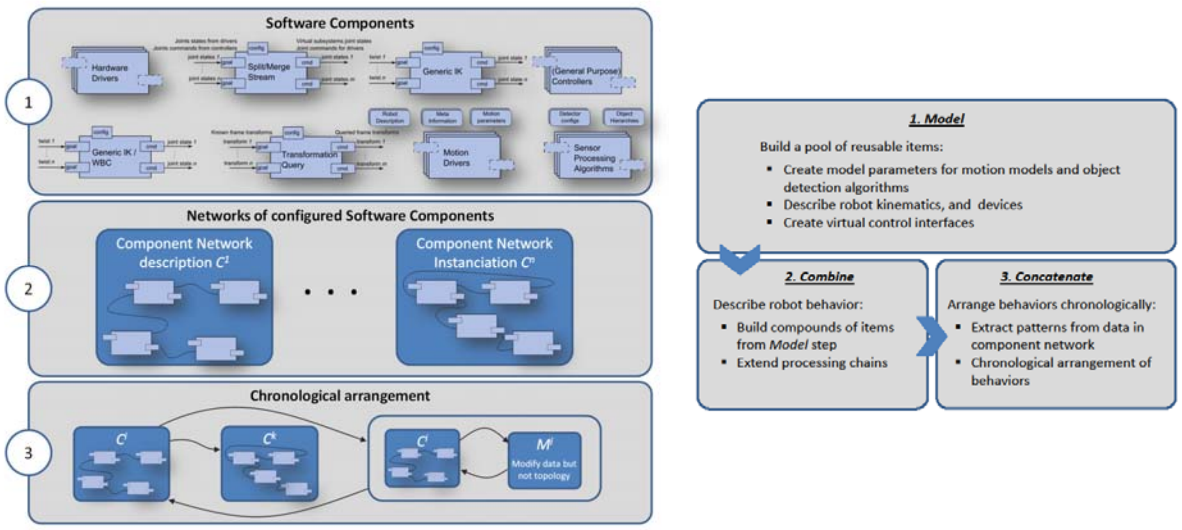

Software Components
Hardware Drivers
Split/Merge Stream
Generic IK
(General Purpose) Controllers
Generic IK / WBC
Transformation Query
Motion Drivers
Sensor Processing Algorithms
config
goal
cmd
1
Networks of configured Software Components
Component Network description C^1
Component Network Instanciation C^n
2
Chronological arrangement
C^i
C^k
C^j
M^j
Modify data but not topology
3
1. Model
Build a pool of reusable items:
Create model parameters for motion models and object detection algorithms
Describe robot kinematics, and devices
Create virtual control interfaces
2. Combine
Describe robot behavior:
Build compounds of items from Model step
Extend processing chains
3. Concatenate
Arrange behaviors chronologically:
Extract patterns from data in component network
Chronological arrangement of behaviors

Figure 2. Workflow to create a specific robot behaviour

Dynamic Control

Many methods to compute the dynamics of a robotic system are based on the availability of the dynamic parameters. In a real case, however, in most cases only imprecise CAD data is available. For that reason, many methods for experimental identification of the dynamic parameters of a robot can be found. However, usually for a small number of joints which are configured in a serial kinematic chain. Our starting point is though a robotic system which might include several arms, a torso, and possibly a mobile base. That is, a significant number of joints and several kinematic chains. For that reason, in this area we developed methods to obtain dynamic models from experimental identification by generating appropriate identification experiments and parameter estimation from the measurements. Furthermore, the methods were expanded to be used on tree-like kinematic structures as for example a dual-arm robot.

A software library was developed which builds first a parametrized dynamic model by using the known robot geometry (see Fig. 3). The next component of the library contains methods for experimental identification of those unknown dynamic parameters. This includes a library to generate identification trajectories (taking into account joint limits, maximum velocities and accelerations, for instance) and a component for the estimation of the parameters from the experimental measurements.

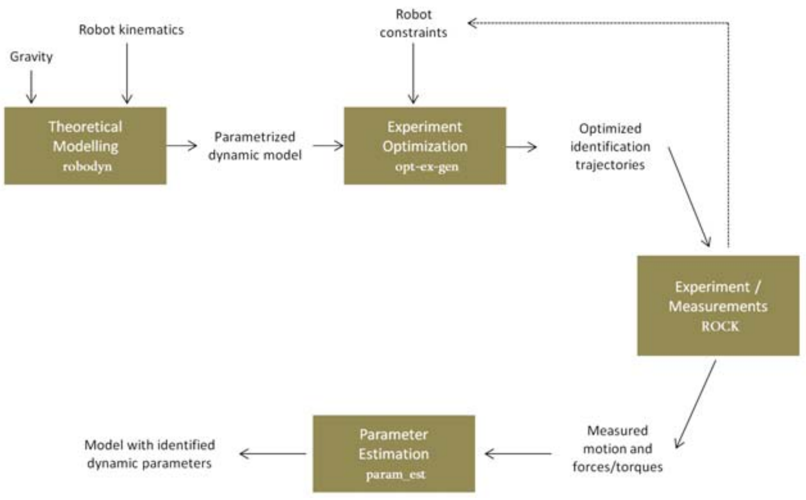
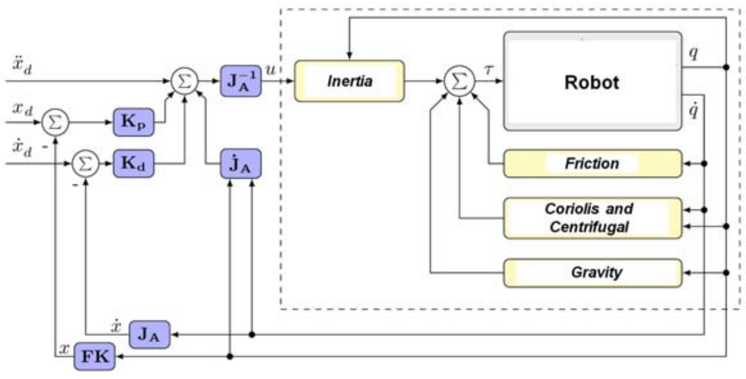

Gravity
Robot kinematics
Theoretical Modelling robodyn
Parametrized dynamic model
Robot constraints
Experiment Optimization opt-ex-gen
Optimized identification trajectories
Experiment / Measurements ROCK
Measured motion and forces/torques
Parameter Estimation param_est
Model with identified dynamic parameters

Figure 3. Software library for identifying the parameters of the robot dynamics model

Once the model is available, the dynamic controller can be implemented (see Fig. 4). The current status of our implementation is divided in a local joint control and a central controller of the whole system (the arm or arms). However, different modes can be selected depending on which computations are done locally or centrally: (1) only local feedback control of position and velocity, in this case, only using a part of the dynamic model for compensating locally for the actuator friction (if desired), (2) local limited feedback control and feedforward control (centrally computed) of the required torques by using the dynamic model, (3) a complete central, model-based dynamic control, locally only having a motor current controller and (4) a complete distributed model-based control in which each joint computes a part of the dynamic model by taking advantage of the recursive Newton-Euler equations and sends its contribution to its adjacent joint. In this case, nothing is computed centrally.



.

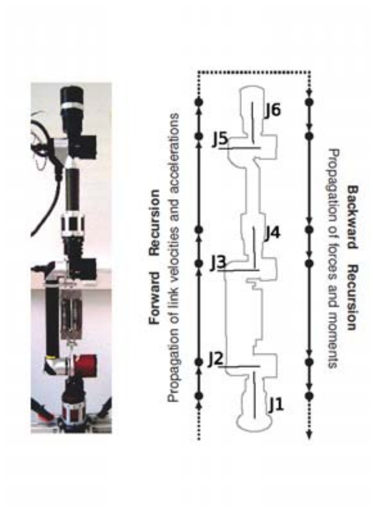

J6
J5
J4
J3
J2
J1
Forward Recursion
Propagation of link velocities and accelerations
Backward Recursion
Propagation of forces and moments

Figure 4. *(left)* Model-based dynamic control, *(right)* 6-DOF robot COMPI is shown together with the use of the recursive nature of the Newton-Euler method for computing the dynamics of the robot in a distributed manner

Trajectory Planning

Various approaches have been suggested for solving a motion planning problem. Artificial potential fields, search based planning, optimization based planning, combinatorial planning and sample based planning are examples of these approaches. Sample based planners are very potent in solving motion planning problems even in high-dimensional spaces. As a disadvantage, the solution usually comes with redundant movements and the executed trajectory on robot looks unnatural. Therefore post processing steps (such path smoothing and optimization) are needed. We have implemented our sample based motion planning library based on OMPL [12] and in a similar fashion as MoveIt! [13] for the main pipeline. Additionally, we have included some functionality for dual-arm operations with Cartesian constraints, self-filtering and the possibility to plan in dynamic environments via replanning.

The main algorithm for planning under Cartesian constraints is based on RRT. Our planner draws random samples in joint space. If any constraints have been set, the planner changes the sampling space from joint space to Cartesian space and plans for the object which is being manipulated. In fact, the drawn sample represents the object position. The upper and lower bounds of sampling space are determined by the imposed constraint.

The imposed Cartesian constraints are expressed similar to the “task space regions” from CBiRRT [14]. After finding the end-effector pose from object pose and grasping pose, we search for inverse kinematic solution. If a solution is found, the joint values are within the joint limits, and robot state is not in collision, the solution will be added to the RRT.

Whole-body reactive control

Whole Body Control (WBC) is an emerging robot control method, which allows to specify and control complex reactive robot motions. Frameworks like iTaSC [15] use geometric constraints to define multi-objective interaction tasks. In the context of the BesMan project we apply a constraint-based Whole Body Control scheme for executing reactive robot motions, incorporating the physical constraints of a system, integrating multiple, disparate sensor-based controllers and optimally utilizing the redundant degrees of freedom in complex systems, like e.g. AILA [16]. In particular we integrated a constraint-based, multi-objective robot controller similar to [15] in our Rock framework (see figure 5) and provided the infrastructure to execute action plans based on motion constraints, as well as change the constraints online. All the active motion constraints and their parametrizations compose a subtask in our action plan that is executed by the robot.

In BesMan, we applied our whole body control approach to multiple robotic systems and in different tasks, for example dual arm manipulation (see Figure 6 and Video [17]).

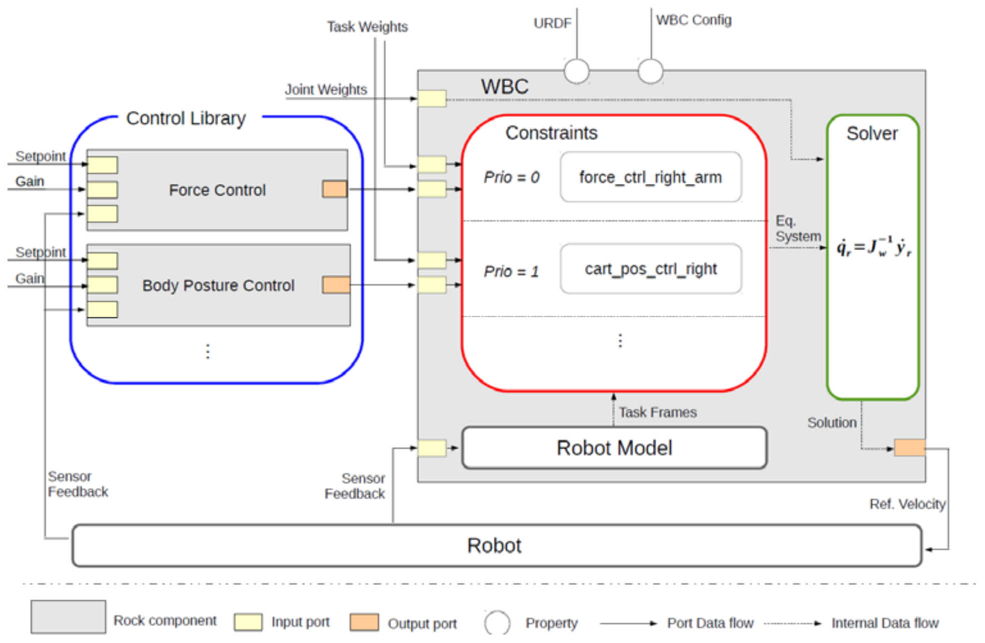


Figure 5. Integration of a whole-body reactive control framework in Rock. We distinguish between a collection of generic, robot independent controllers (control library) and the actual WBC. Similar as in [15], each controller feeds a constraint in WBC. Each constraint can be parametrized by its priority and task weights.

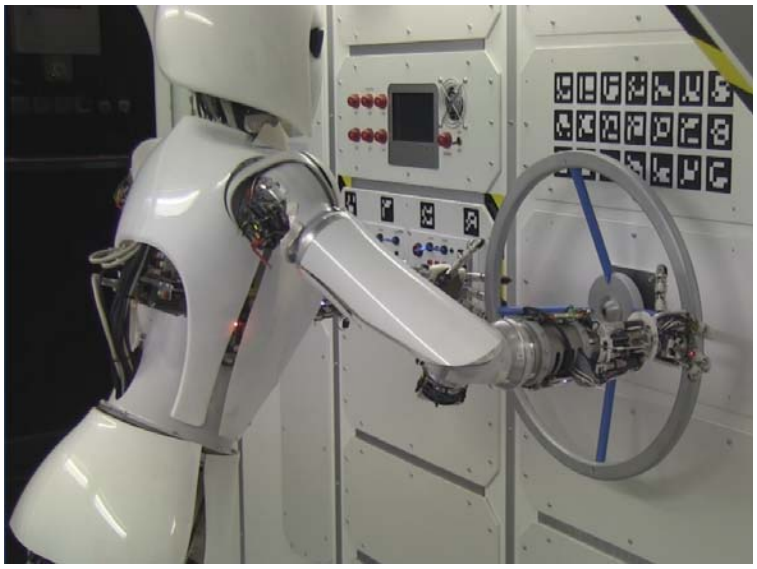


Figure 6. AILA turning a handwheel and making use, among others, of the whole-body control component

CONCLUSIONS

This paper presented ongoing work towards creating a software architecture centered on the concept of reusing software components to create different robot behaviours. The robot architecture foresees a behaviour template pool which stores templates to be instantiated with the current contextual information to create a certain robot behavior. Furthermore, the architecture integrates and makes use of three key main components for a flexible and reliable robot manipulator under uncertain or dynamic environments: a whole-body reactive control module, a (dynamic) trajectory planning and the use of the robot dynamics within the low-level control.

ACKNOWLEDGEMENT

This work was supported through two grants of the German Federal Ministry of Economics and Technology (BMWi, FKZ 50 RA 1216 and FKZ 50 RA 1217)

REFERENCES

[1] Okamura, A.M. and Smaby, N. and Cutkosky, M. R. *An overview of dexterous manipulation*. Proceedings of IEEE International Conference on Robotics and Automation (ICRA), 2000

[2] S. Chitta, B. Cohen, and M. Likhachev. *Planning for autonomous door opening with a mobile manipulator.* Proc. of IEEE International Conference on Robotics and Automation (ICRA), 2010.

[3] S. Srinivasa, D. Ferguson, C. Helfrich, D. Berenson, A. Collet, R. Diankov, G. Gallagher, G. Hollinger, J. Kuffner, and M. VandeWeghe. *HERB: a home exploring robotic butler.* Autonomous Robots, 2009.

[4] M. Fuchs, C. Borst, P. R. Giordano, A. Baumann, E. Kraemer, J. Langwald, R. Gruber, N. Seitz, G. Plank, K. Kunze, R. Burger, F. Schmidt, T. Wimboeck, and G. Hirzinger, *"Rollin' justin – design con-*

siderations and realization of a mobile platform for a humanoid upper body. Proceedings of the 2009 IEEE international conference on Robotics and Automation. Piscataway, NJ, USA: IEEE Press, 2009, pp. 1789–1795.

[5] D. Katz, E. Horrell, O. Yang, B. Burns, T. Buckley, A. Grishkan, V. Zhylkovskyy, O. Brock, and E. Learned-Miller. *The UMass mobile manipulator UMan: An experimental platform for autonomous mobile manipulation.* In Workshop on Manipulation in Human Environments at Robotics: Science and Systems, 2006.

[6] Website Project BesMan at DFKI Robotics Innovation Center. http://robotik.dfki-bremen.de/en/research/projects/besman.html (visited on 09.09.2015)

[7] Jan Hendrik Metzen, Alexander Fabisch, Lisa Senger, José de Gea Fernández, Elsa Andrea Kirchner. *Towards Learning of Generic Skills for Robotic Manipulation.* In KI - Künstliche Intelligenz, German Journal on Artificial Intelligence, Springer, volume 28, number 1, pages 15-20, 2014

[8] Flanagan, J. R. and Bowman, M. C. and Johansson, R. S. *Control strategies in object manipulation tasks*. Current Opinion in Neurobiology, Vol. 16, pp. 650—659, December 2006

[9] ROCK Website: http://www.rock-robotics.org/stable/ (visited on 09.09.2015)

[10] Sylvain Joyeux and Frank Kirchner and Simon Lacroix. *Managing plans: Integrating deliberation and reactive execution schemes.* Robotics and Autonomous Systems. Vol. 58, pp. 1057-1066, 2010

[11] Malte Wirkus. *Towards Robot-independent Manipulation Behavior Description*. In Proceedings of the 5th International Workshop on Domain-Specific Languages and models for ROBotic systems, (DSLRob), 2014

[12] Ioan A. Sucan and Mark Moll and Lydia E. Kavraki. *The Open Motion Planning Library*. IEEE Robotics & Automation Magazine, Vol. 19, pp. 72-82, December 2012

[13] Ioan A. Sucan and Sachin Chitta, "*MoveIt!*", http://moveit.ros.org (visited on 09.09.2015)

[14] Dmitry Berenson, Siddhartha Srinivasa, David Ferguson, and James Kuffner. *Manipulation Planning on Constraint Manifolds*. IEEE International Conference on Robotics and Automation (ICRA '09), May, 2009.

[15] Smits, R., De Laet, T., Claes, K., Bruyninckx, H., De Schutter, J. *iTASC: a tool for multi-sensor integration in robot manipulation*. In : Proceedings of the Multisensor fusion and integration for intelligent systems, Seoul, Korea, Aug 20-22, 2008

[16] Johannes Lemburg, José de Gea Fernández, Markus Eich, Dennis Mronga, Peter Kampmann, Andreas Vogt, Achint Aggarwal, Yuping Shi, Frank Kirchner. In Proceedings of IEEE International Conference on Robotics and Automation, (ICRA-11), 09.5.-15.5.2011, Shanghai, o.A., pages 5147-5153, May/2011. ISBN: 978-1-61284-380-3.

[17] DFKI Youtube Channel. BesMan first official demonstration using the robot AILA: https://youtu.be/f8p5MPzvcyY (visited on 30.09.2015)