German Research Center for Artificial Intelligence

Research Report RR-13-03

Motion Control of the Humanoid Robot AILA

Dennis Mronga and **Achint Aggarwal**, 03/2013

Research Report RR-13-03

German Research Center for Artificial Intelligence (DFKI) GmbH

Bibliographic information published by the German National Library

The German National Library lists this publication in the German National Biography; detailed bibliographic data are available in the internet at http://dnb.ddb.de.

Issue RR-13-03 (2013)
ISSN 0946-008x

German Research Center for Artificial Intelligence
Deutsches Forschungszentrum für Künstliche Intelligenz
DFKI GmbH

Founded in 1988, DFKI today is one of the largest nonprofit contract research institutes in the field of innovative software technology based on Artificial Intelligence (AI) methods. DFKI is focusing on the complete cycle of innovation – from world-class basic research and technology development through leading-edge demonstrators and prototypes to product functions and commercialization.

Based in Kaiserslautern, Saarbrücken and Bremen, the German Research Center for Artificial Intelligence ranks among the important 'Centers of Excellence' worldwide. An important element of DFKI's mission is to move innovations as quickly as possible from the lab into the marketplace. Only by maintaining research projects at the forefront of science DFKI has the strength to meet its technology transfer goals.

The key directors of DFKI are Prof. Wolfgang Wahlster (CEO) and Dr. Walter Olthoff (CFO). DFKI's research departments are directed by internationally recognized research scientists:

- Knowledge Management (Prof. A. Dengel)
- Cyber-Physical Systems (Prof. R. Drechsler)
- Robotics Innovation Center (Prof. F. Kirchner)
- Innovative Retail Laboratory (Prof. A. Krüger)
- Institute for Information Systems (Prof. P. Loos)
- Embedded Intelligence (Prof. P. Lukowicz)
- Agents and Simulated Reality (Prof. P. Slusallek)
- Augmented Vision (Prof. D. Stricker)
- Language Technology (Prof. H. Uszkoreit)
- Intelligent User Interfaces (Prof. W. Wahlster)
- Innovative Factory Systems (Prof. D. Zühlke)

In this series, DFKI publishes research reports, technical memos, documents (eg. workshop proceedings), and final project reports. The aim is to make new results, ideas, and software available as quickly as possible.

Prof. Wolfgang Wahlster
Director

Motion Control of the Humanoid Robot AILA

Dennis Mronga and **Achint Aggarwal**

03/2013

Research Report RR-13-03 des
Deutschen Forschungszentrums für Künstliche Intelligenz (DFKI)

Abstract

This technical report describes ongoing work regarding motion control of the humanoid robot AILA. Recently, software components for motion control and planning, sensor based control, telemanipulation and task level control have been developed for the system. This paper gives an overview on algorithms, methods and control structures used, demonstrates their feasibility by the means of experimental results and summarizes the remaining issues and shortcomings.

Zusammenfassung

Dieser technische Bericht beschreibt die laufende Arbeiten bezüglich der Bewegungsteuerung des humanoiden Roboters AILA. In letzter Zeit wurden Softwarekomponenten bezüglich Bewegungssteuerung- und -planung, sensorbasierter Regelung, Telemanipulation und Task-Level-Steuerung für das System entwickelt. Dieser Bericht soll einen Überblick über Algorithmen, Methoden und Steuerstrukturen geben, experimentelle Ergebnisse darstellen, sowie die verbleibenden Probleme und Schwachstellen zusammenfassen.

Contents

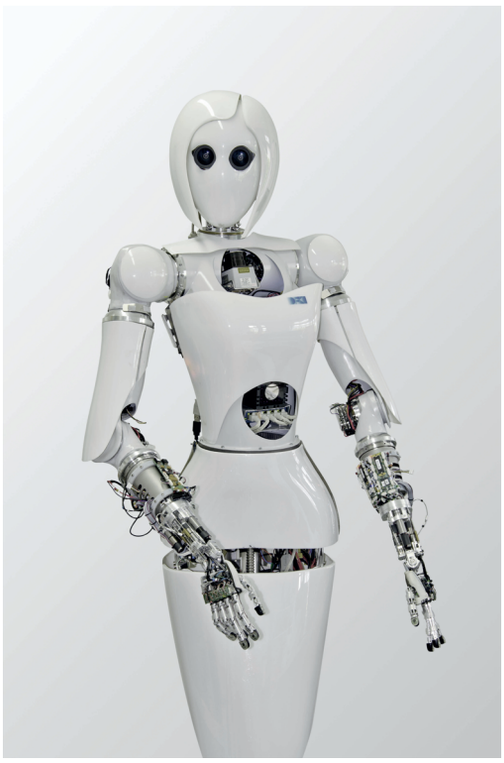


Figure 1: Mobile Dual-Arm System AILA [11]

1 Introduction and System Overview

Although many robotic systems have nowadays reached a level of maturity which allows robust task execution even in unstructured environments, the area of mobile manipulation remains an active area of research. Humanoids pose a particular challenge regarding control due to their high number of degrees of freedom that have to be coordinated and the required integration of multiple sensing modalities. Examples for humanoid robots are DLR's Rollin Justin [4], the HPR-2 robot [8], Armar-III [1] and Honda's Asimo [16]. Key software components for the control of such a system are object recognition, motion control and planning, force control, navigation and coordination on task level.

The mobile, dual arm robot AILA (Figure 1) has first been described in [11]. It consists of a 12 DOF mobile base and a humanoid upper body with 4 DOF in the torso, 7 DOF in each arm and 2 DOF in the pan-tilt unit of the head. Additionally, two five-fingered hands are integrated in the system to enable dexterous manipulation skills. On the sensor side the system integrates, amongst others, a stereo camera system in the head and a force sensor in each forearm. Regarding motion control of AILA, a large number of DOF have to be controlled and coordinated, while reacting to sensory input from multiple, disparate sources.

In this technical report, we give an overview on our ongoing efforts regarding the development of software components, including vision, force and motion control, motion planning and task level coordination, and their integration in the humanoid robot AILA. The focus is laid on the components that are required for manipulation tasks, while mobility shall not be considered here. In addition to the components that are used for autonomous task execution like object recognition and motion planning, we also describe our work on telemanipulation.

This document is structured as follows. Section 2 gives an architectural overview, while sections 3 to 7 describe our work on motion control, vision, motion planning, sensor based control, telemanipulation and task level control, respectively. In section 8 some experimental results are illustrated. Finally, section 9, discusses remaining issues and future work.

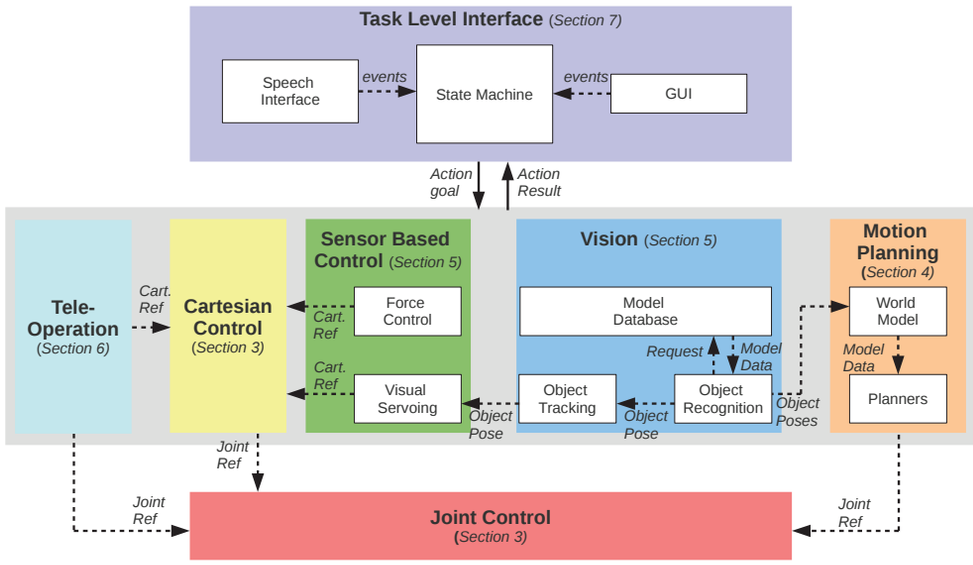


Figure 2: Overview of the software components and the major data flow between them. Dashed lines represent topic data streams and continous lines action goals/results.

2 Architectural Overview

The software architecture of AILA is based on the ROS framework [15], which mainly serves as communication layer between different processes. ROS uses *topics* for datastream-like, high bandwith communication, *services* for request-reply interaction and *actions* in order to trigger more complex tasks that might require a long time to process and thus should provide feedback about the current task state. Figure 2 gives an architectural overview of the software used in AILA and the major data ports. On task level, we use a GUI or speech command input to create events, which trigger state transitions in an event based finite state machine (see section 7). This state machine monitors the execution of different subtasks, e.g. to detect an object, perform visual servoing to a certain position or to close a grasp. Here, two ways of solving a task are possible and shall be highlighted within this report: The reactive, sensor-based approach (see section 5), which uses for example Visual Servoing and force control for task execution, and the motion planning based approach (see section 4), which uses a static world model to generate motion plans. In the end, both approaches generate position or velocity based joint trajectories, which are sent to the joint control interface, which is illustrated in section 3. Furthermore, we give an overview of our work on teleoperation in section 6.

3 Motion Control

Motion control in robotics deals with the generation of robot motor commands that guarantee the execution of a specified task in free space, e.g. following a predefined trajectory. Regarding the design of the motion control components for the robotic system AILA, there were four basic problems:

- *Motion Generation:* The trajectories applied to the joints should be smooth and dynamically feasible
- *Generality:* The control software should be generally applicable to other robotic systems
- *Consistency:* All 50 dof of the robot should be controllable with the same interface
- *Safety:* The robot should be protected from self collisions, infeasible user input and other damages

The following sections give a general overview of the motion control software of AILA and describe how we solved these problems.

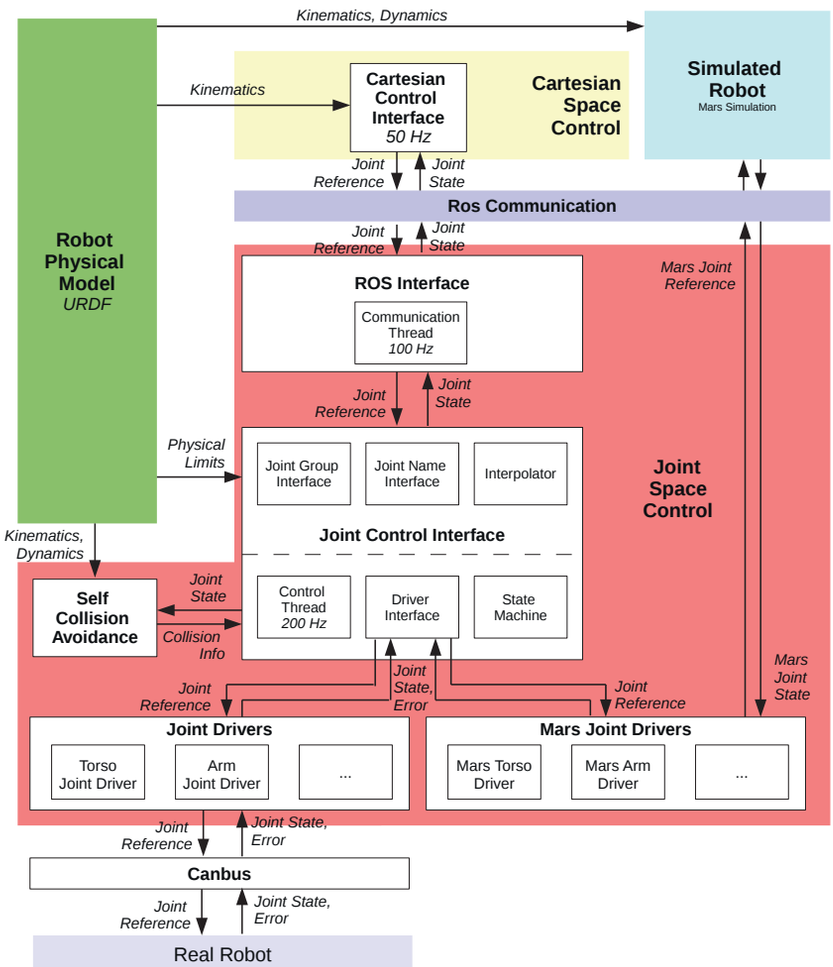


Figure 3: General Overview over the Motion Control Components in AILA

3.1 Overview

The low level motion control in AILA is performed in a decentralized manner. Each motor is equipped with an electronic stack, which contains a micro-controller/FPGA. The idea is that the micro-controller/FPGA performs the motor control with high frame rates, while the central CPU sends reference values with lower frame rates. The general concept of a distributed control structure is decribed in detail in [2]. Such a setup has many advantages, e.g. yielding highest control rates, parallel processing of motor signals and hiding the implementation of low level motion control from the high level programmer. However, it requires a sophisticated software control interface, that collects and synchronizes the signals from all motors. A further problem is that heterogeneous controller concepts are used in AILA. For example the arm joints are controlled by FPGA, while the head and fingers are accessed via servo motors with integrated controllers designed by the manufacturer. This makes it difficult to write common software interfaces for all joints, since the different controllers may have heterogeneous parameters, functionality and characteristics.

Figure 3 gives a general overview of the components and the flow of information in the motion control of AILA. The individual components shall be explained in the following.

Joint Control Interface The Joint Control Interface is the central component in the joint control of AILA and has the following tasks:

- Provide the user interface for joint control and hide the driver functionality from the user
- Observe the system state on joint level and apply safety measures
- Perform trajectory interpolation

From the higher level, the user can access the joints either via their joint names (*Joint Name Interface*) or, for convenience, via joint groups (*Joint Group Interface*), that can be arbitrarily defined. Joint groups might be for example "torso", "body" or "head". Currently, position and velocity control is supported. If a new position or velocity reference is set by the user, the *Interpolator* will dynamically generate a whole body trajectory (see next section), which is sent to the *Driver Interface*. The interface to the drivers is fully hidden from the user, which is indicated by the dashed line within the Joint Control Interface component in figure 3. The *Control thread* runs at a frequency of 200 Hz. In each control cycle it reads the current joint state from the drivers, checks if the Interpolator provided a new sample, sends the values to the correct drivers and checks for error messages from driver level. The *State Machine* primarily makes sure that initialization and shut down of the Joint Control Interface happen correctly.

ROS Interface The ROS interface runs the communication thread, which sends joint data via ROS to other AILA components and receives commands via ROS topics. Each joint group has two topics, one for receiving joint references and one for sending joint data. Furthermore, debug data is sent on a separate channel. Joint parameters like maximum velocity or the currently used interpolation method are propagated to other AILA components via the ROS parameter server.

Joint Drivers Each joint driver inherits basic functionality from a Joint Driver Template, so that all drivers have the same interface. They are collected in the Driver Interface of the Joint Control Interface component and provide access to the Low-Level controllers at the joints of AILA via CAN-Bus.

Mars Joint Drivers For simulation of AILA, we use the MARS environment[1], which has been developed at DFKI. The Mars Joint Drivers inherit from the same template as the real drivers, so that the interface for both, the real robot and the Mars simulation is the same. Like this, we can switch from the real system to the simulation just by changing a parameter at start-up of the Joint Control Interface and test all the higher level functionality under most realistic conditions.

Robot Physical Model The kinematic model of AILA has been described in the URDF (Unified Robot Description Format-Language[2]), which is the native modeling format of ROS. URDF is an XML-markup language, which supports kinematics, dynamics and physical constraints. The robot model is used for initialization of the Mars model, the Cartesian Control, the Self Collision Avoidance and the Joint Control Interface.

Cartesian Control Interface Provides a Cartesian Control Interface using the kinematic model of AILA (see section 3.2).

Self Collision Avoidance Checks for self collisions of e.g. AILA's arms with the body (see section 3.3).

3.1.1 Motion Generation

Interpolation is a base requirement in order to provide a smooth and visually appealing robot motion. Furthermore, the generated trajectories have to be dynamically feasible, e.g. maximum joint velocities, accelerations and torques must not be exceeded in order to enable the robot to follow the desired motion and prevent damaging the system. In the robot AILA different interpolation methods have been implemented and shall be described in the following section.

If a new joint space reference is sent to the Joint Control Interface component, the Interpolator will provide a trajectory with N samples as follows:

$$q(t_k) = a_5 t_k^5 + a_4 t_k^4 + a_3 t_k^3 + a_2 t_k^2 + a_1 t_k + a_0 \quad k = \{0, ..., N\}, \quad t_k = \{0, ..., 1\} \tag{1}$$

In AILA, four different motion generation methods are implemented and the coefficients a_i are computed according to the chosen method, as shown in table 3.1.1.

These schemes can be applied to both, position and velocity control and ensure smooth trajectories. The interpolation has been implemented to work online, which means that the currently executed trajectory can be modified on the fly. If a new reference is sent to the Joint Control Interface while a previous trajectory

[1] http://www.gitorious.org/rock-simulation/mars
[2] http://www.ros.org/wiki/urdf, accessed 2012/12/08

	a_0	a_1	a_2	a_3	a_4	a_5
None	q_N	0	0	0	0	0
Linear	q_0	$q_N - q_0$	0	0	0	0
Cubic	q_0	$\dot{q}_0$	$-3q_0 + 3q_N - 2\dot{q}_0 - \dot{q}_N$	$2q_0 - 2q_N + \dot{q}_0 + \dot{q}_N$	0	0
Quintic	q_0	$\dot{q}_0$	$0.5\ddot{q}_0$	$(-20q_0 + 20q_N - 3\ddot{q}_0 + \ddot{q}_N - 12\dot{q}_0 - 8\dot{q}_N)/2$	$(30q_0 - 30q_N + 3\ddot{q}_0 - 2\ddot{q}_N + 16 * \dot{q}_0 + 14\dot{q}_N)/2$	$(-12q_0 + 12q_N - \ddot{q}_0 + \ddot{q}_N - 6\dot{q}_0 - 6\dot{q}_N)/2$

Table 1: Interpolation coefficients for the four different interpolation methods. Here $q_0 = q(t_0)$ and $q_N = q(t_N)$ are the current and the reference value (initial and end value of the interpolated trajectory), while $\dot{q}_0 = \dot{q}(t_0)$ and $\dot{q}_N = \dot{q}(t_N)$ describe their first and $\ddot{q}_0 = \ddot{q}(t_0)$ and $\ddot{q}_N = \ddot{q}(t_N)$ their second derivatives, respectively.

has not yet been finished, the spline interpolator will take into account the current position, velocity and acceleration in order to generate a smooth transition. The effect of the different interpolation methods can be seen in figure 4. Obviously, the quintic interpolation provides the smoothest motions, since it is based on polynomials of higher order than the other methods. However, it requires the current acceleration $\ddot{q}_0$, which is not available in AILA as measurement. Thus, it must be computed as derivative from the velocity measurement, which gives very noisy results and may lead to non-continuous trajectories. Therefore, cubic interpolation is chosen as the preferred method for motion generation.

3.1.2 Control Parameter Identification

Due to the high gear ratio used in the joints of AILA (1:100 - 1:120), the control of the joints can be considered as decoupled, which means that the motion of one joint does not influence the behavior of the others significantly. Therefore, simple PID controllers, which control each joint separately without considering robot dynamics or non-linear coupling effects, are already sufficient to provide relatively good trajectory tracking behavior. Prerequisite is an identification of optimal PID parameters.

The control parameter identification is based on the step response of each joint of the robot. Relying on the scientific literature about motion control as in [14, 9], one can assume that the dynamics of the brushless DC motors including the gears used in AILA can be described as follows:

$$G(s) = \frac{\Theta(s)}{I(s)} = \frac{A}{s \cdot (s + B)} \tag{2}$$

$$A = \frac{K_t}{J \cdot \eta}, \quad B = \frac{\nu}{J} \tag{3}$$

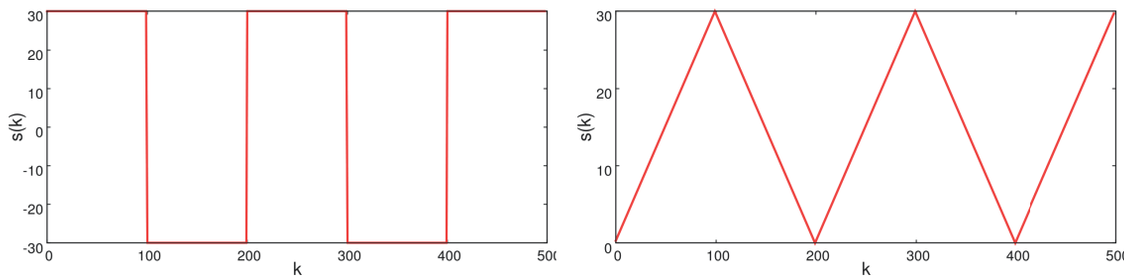
where $G(s)$ is the transfer function[3] of the system, $\Theta(s)$ and $I(s)$ the motor angle and current, respectively, K_t the motor torque constant, ν the viscous friction, J the motor's rotor inertia and η the gear ratio.

With the step response of the motor, obtained when using the lowest possible control gain, the parameters A and B can be computed with the help Matlab's System Identification Toolbox and Simulink. Now, a simple PD-control scheme is applied, where the controller parameters can be computed as follows:

$$K_p = \frac{3 \cdot p^2}{A}, \quad K_D = \frac{3 \cdot p - B}{A} \tag{4}$$

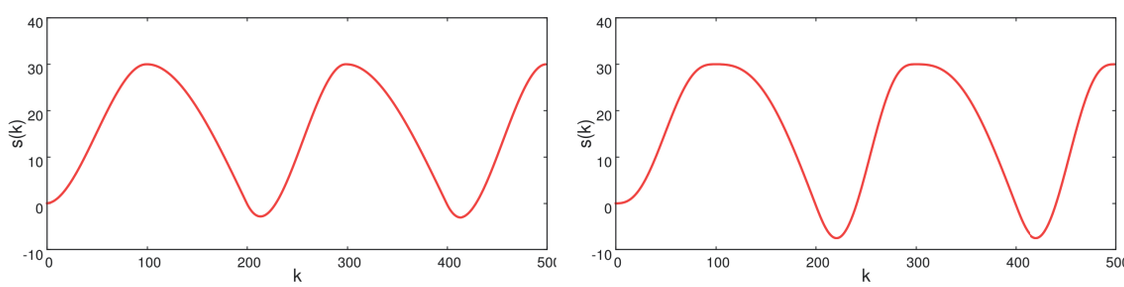
where p is the position of the system's poles on the real axes of the pole-zero plot of the overall transfer function. If the pole is chosen to be far from the imaginary axis, the system response will be very fast, but also the system may become unstable more easily. A pole that is chosen close to the imaginary axis leads to a slow response, but also higher stability. The maximum value of the poles depends, amongst others, on the sample rate of the controller. In case of AILA we chose a high control frequency (10 Khz), so we can choose

[3]The transfer function $G(s)$ describes the relationship between input and output of linear, time-invariant system in the frequency domain, where s is the temporal or spatial frequency of a system [13]



(a) No interpolation: The generated trajectory is equal to the reference value

(b) Linear interpolation



(c) Cubic interpolation

(d) Quintic interpolation

Figure 4: Interpolation methods for motion generation: The reference value $s(k)$ (y-axis), which represents the joint angle of one particular joint, is switched from -30° to 30° at certain intervals. The trajectory generator transforms these references into smooth motions.

poles far from the imaginary axis. However, while testing this approach with different values of p, it has been observed that higher values lead to stronger vibrations, but also to more accurate trajectory tracking. Therefore, a compromise between smoothness of the arm motion and the precision of trajectory following is required. The results can be seen in figure 5. It is visible that, when using higher values of p, the trajectory can be tracked more accurately. However, also the vibrations have been stronger for the experiment with $p = 40$.

3.2 Cartesian Control

The *Cartesian Control Interface*, as shown in figure 3, has access to the kinematic model of AILA. The advantage of URDF as a robot modeling format is that it nicely integrates with both, ROS and KDL (Kinematics and Dynamics Library[4]). The latter is an OROCOS library that provides description of kinematic chains and trees, as well as forward and inverse kinematics computation. Based on the weighted damped least squares (WDLS) solver from KDL, an additional velocity solver has been implemented. It uses an adaptive damping factor in order to better avoid singularities than the WDLS solver and a nullspace projection method in order to include the physical constraints of the robot in the solution generation (as in [17]). The damped least squares solution is generated as follows:

$$\dot{\mathbf{q}} = \mathbf{J}^* \mathbf{v}_e \tag{5}$$

with

$$\mathbf{J}^* = \mathbf{J}^T (\mathbf{J}\mathbf{J}^T + k^2 \mathbf{I})^{-1} \tag{6}$$

Here, $\mathbf{J} = \mathbf{J}(\mathbf{q})$ is the current Jacobian of the robot, $\mathbf{v}_e$ is the desired end effector velocity in Cartesian space, $\dot{\mathbf{q}}$ is the desired joint velocity, I is the identity matrix and k is an adaptive damping factor, which is

[4] http://www.orocos.org/kdl. Accessed 2012/11/25

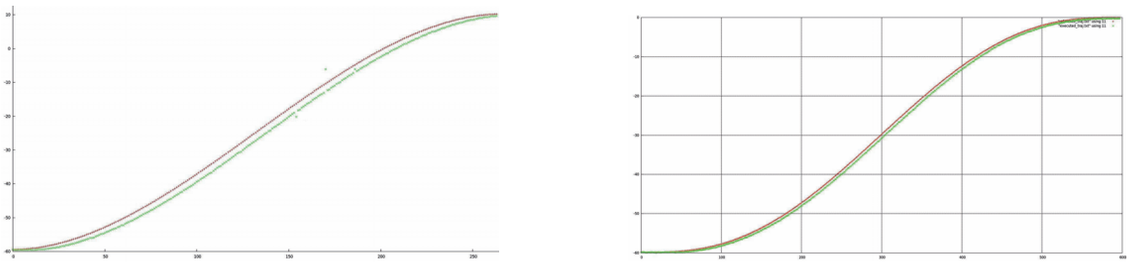


Figure 5: Reference (red) and executed (green) trajectory for one joint (y-axis, in degrees) with respect to time (x-axis), *Left:* $p = 30$, *Right:* $p = 40$

modified with respect to the current manipulability[5] M of the robot:

$$k = k_0 \cdot (1 - \frac{M}{M_0}) \tag{7}$$

with

$$M = \sqrt{|\mathbf{J}\mathbf{J}^T|} \tag{8}$$

Here M_0 is the maximum manipulability of the robot, which has been determined experimentally and k_0 is the maximum damping factor. If M approaches M_0 the damping factor converges to zero and the "normal" least squares solution is applied. If the system is close to a singular configuration, the damping factor converges to k_0. Thus, the system is being guided safely through the singularity, while following the reference Cartesian velocity with reduced accuracy.

Additionally, a nullspace projection is used in order to solve a secondary task in the nullspace of the primary task and exploit the redundancy of the robot:

$$\dot{\mathbf{q}} = \mathbf{J}^* \mathbf{v}_e + (\mathbf{I} - \mathbf{J}^* \mathbf{J}) \dot{\mathbf{q}}_0 \tag{9}$$

The matrix $(\mathbf{I} - \mathbf{J}^* \mathbf{J})$ provides the projection on the nullspace of the Jacobian $\mathbf{J}$. The nullspace velocity $\dot{\mathbf{q}}_\mathbf{0}$ can be chosen in order to satisfy an arbitrary constraint. A typical choise is

$$\dot{\mathbf{q}}_0 = \lambda (\frac{\delta w(\mathbf{q})}{\delta \mathbf{q}})^T \tag{10}$$

where $w(\mathbf{q})$ is an objective function of the joint angles and λ is the nullspace gain. Since the solution moves along the gradient of the objective function, it attempts to maximize it locally, while complying with the primary objective. The object function can be chosen, for example, to achieve joint limit avoidance:

$$w(\mathbf{q}) = -\frac{1}{2N} \sum_{i=1}^{N} (\frac{q_i - \bar{q}_i}{q_{i,max} - q_{i,min}})^2 \tag{11}$$

Here, the constraint term $w(\mathbf{q})$ describes the distance from the joint limits $q_{i,max}$ and $q_{i,min}$. While providing a homogeneous solution for $\dot{\mathbf{q}}$ that ensures execution of $\mathbf{v}_e$ in task space, this method also produces internal motions that try to optimize $\mathbf{q}$ with respect to $\mathbf{w}(\mathbf{q})$.

3.3 Safety Measures

In order to protect the system from damage, a safety concept has been integrated with AILA, including self-collision avoidance and a couple of low level safety measures. The self-collision avoidance is based on the work described in [19]. It has been adapted to AILA and integrated with the Joint Control Interface component (see figure 3). The algorithm uses the current joint state as input and determines swept volumes for each link of the system (see figure 6). These volumes grow and shrink with respect to the current joint velocity. They determine how far a link is located from a possible collision and, given the maximum deceleration of the

[5]The manipulability is a measure that describes the distance of a robot configuration to a singularity. The lower the manipulability, the lower the distance to a singular configuration of the robot.

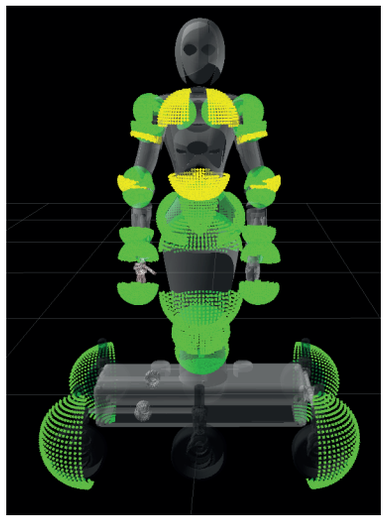


Figure 6: Swept volumes used for collision avoidance

system, communicate to the Joint Control Interface if a joint has to brake. Therefore, in each control step, the current joint state is passed by Joint Control Interface to the Self Collision Avoidance and a collision info is returned, which can be used to modify the current reference value. The current reference is set to zero if a collision is possible.

Further safety measures are implemented on the lower levels and consider the joint limits, misuse by the user and possible CAN communication errors. If a safety constraint is violated, an error code is sent from micro-controller/FPGA to the Joint Control Interface and the motors are switched off immediately.

4 Motion Planning

Motion planning refers to the process of describing the detailed motions of all the joints of the robot in a coordinated manner to enable it to execute some meaningful task. Motion planning for AILA consists of planning the coordinated movements of 18 joints (two arms each with seven joints and four torso joints. The mobile base and the multifingered hands are excluded from planning. Instead, objects are grasped with flat palms as shown in [11]), such that the robot can perform the required task without colliding with environment obstacles. The use of the four torso joints along with the two arms provides additional flexibility in operation and increases the reachable space of the robot. Planning in AILA is, for now, done with the assumption of a static environment. Dynamic changes in the environment can be treated by using reactive control concepts as explained in section 5. However, to this point, reactions to unforeseen sensory events do not trigger dynamic re-planning.

In this section we will present the software components for motion planning in AILA. For manipulating e.g. large boxes, it is preferable to have the robot grasp the objects using both hands. This imposes additional planning criteria, like maintaining the relative distance between the hands to make sure the object is not dropped, and maintaining the orientation of certain objects like open or liquid-filled containers.

A typical task for motion planning consists of grasping an object from a given position with respect to the robot (attained by vision), and moving it safely to another position in the environment. This objective involves determining suitable grasping points on the object, possible robot configurations, collision free path planning to attain the grasping configuration, dual arm manipulation of the object to the desired position, and path planning for release of the object. The complexity of these tasks and their interface requires a fault management architecture which is discussed in the following sections along with the other path planning modules.

4.1 Inverse Kinematics Solvers

Joint space path planning is most efficient for high DOF robots. However, tasks are normally defined in Cartesian space. This requires a two-way mapping between the Cartesian and joint spaces. While the

mapping from joint space to Cartesian space is relatively easy for serial chain robots (Forward Kinematics), the inverse mapping is complex given the high redundancy of AILA. We have developed a variety of inverse kinematics solvers for AILA for different applications.

4.1.1 Single arm IK Solver

Closed form IK solvers are generated for each of the AILA arms using IKFast [6]. IKFast contains implementations of several state of the art serial chain IK solvers. It tries to generate solutions for a robot geometry using these solvers sequentially until a solution is found. The IK solvers for arms of AILA are able to find IK solutions in a few microseconds.

4.1.2 Dual arm IK Solver

This solver is responsible for generating collision free joint solutions for both arms of AILA together given their required end effector configurations. Based on the single arm IK solver, all possible self-collision free IK solutions are calculated for each arm by removing the other arm from collision checking. All combinations of the solution sets are then tested for self collisions until a collision free solution is found.

4.1.3 Whole body IK Solver

This solver deals with the generation of collision free joint solutions for all the 18 joints of AILA, given the required end-effector configurations of the arms. Two variations of the whole body IK solver have been implemented. The first is a Gaussian sampling based solver. A torso configuration is sampled using Gaussian sampling about the current torso configuration. The standard deviation for the sampling is a function of the time already elapsed in finding the solutions. For a sampled torso configuration, the dual arm IK solver is used to find solutions for the two arms. This is suitable for determining a solution close to current position. The second variation uses random sampling of the entire torso reachability-space to generate a torso configuration. This is used to quickly determine a reachable configuration of a randomly placed object.

4.2 Motion planning for Dual Arm Grasping and Manipulation

Given a task to grasp and move an object localized with respect to the robot, the first step is to determine an appropriate grasping configuration.

4.2.1 Generating a Grasp Configuration

We limit our scope to deal with non-deformable, rigid, and box-shaped objects. The side faces of the object are determined using the object pose and grasping configurations with both hands parallel to these faces are evaluated. The hand centers are required to coincide with the centers of these faces. However, the hands can be aligned at any angle about the horizontal axis of the object. The full body IK solver is used to generate a collision free configuration for the whole robot. Various rotations of the hands about the horizontal axis are evaluated until a solution is found. If a solution is not found after trying all combinations of angles, the base is moved to a better location with respect to the object.

4.2.2 Grasp Planning

Dual arm grasp planning deals with planning of collision free paths for moving the 18 joints from their current position to the grasp configuration. The bi-directional Rapidly Exploring Random Trees(RRT) based RRT-CONNECT planner [10] is used to find a collision free path.

RRT is an exploration algorithm for quickly searching high-dimensional spaces that have both global constraints (arising from workspace obstacles and velocity bounds) and differential constraints (arising from kinematics and dynamics). It is an incremental search method in which a tree (or two trees in the bidirectional version) is grown incrementally from the initial robot state by adding a new edge and vertex in each iteration after performing some local motion. The key idea is to bias the exploration toward unexplored portions of the space by randomly sampling points in the state space, and incrementally pulling the search tree toward them. Since the complexity of motion planning problems increases exponentially with the state-space dimensions, randomized planning approaches enable solving these challenging problems efficiently, at the expense of not being able to guarantee that a solution will be found in finite time. The resulting method is much more efficient than brute-force exploration of the state space.

Algorithm 1 $BUILD\_RRT(x_{init})$ [5]

$T.init(x_{init})$
$for\ k = 1\ to\ K\ do$
  $EXTEND(T, x_{rand})$
$return\ T$

Algorithm 2 $EXTEND\_RRT(T, x)$ [5]

$x_{near} \leftarrow NEAREST\_NEIGHBOR(x, T)$
if $NEW\_STATE(x, x_{near}, x_{new}, u_{new})$ **then**
  $T.add\_vertex(x_{new})$
  $T.add\_edge(x_{near}, x_{new}, u_{new})$
end if

The basic RRT construction algorithm is given in Algorithm 1 and Algorithm 2. A simple iteration is performed in which each step attempts to extend the RRT by adding a new vertex that is biased by a randomly selected state, $x \in X$. The EXTEND function selects the nearest vertex already in the RRT to x. The function NEW STATE makes a motion toward x by applying an input $u \in U$ for some time increment Δt. This input can be chosen at random, or selected by trying all possible inputs and choosing the one that yields a new state as close as possible to the sample, x. NEW STATE is also a state qualification function that implicitly uses the collision detection function to determine whether the new state (and all intermediate states) satisfy the global constraints. For many problems, this can be performed quickly (in almost constant time). If NEW STATE is successful, the new state and input are represented in x_{new} and u_{new} , respectively. Figure 7 shows an RRT grown from the center of a square region in the plane. In this example, there are no differential constraints (motion in any direction is possible from any point).

In the bi-directional variant of the RRT, in addition to growing a tree from the starting state, a second tree is also grown from the goal state. Such trees grow in four steps:

1. Grow start-tree towards a random unexplored configuration.
2. Grow goal-tree towards a random unexplored configuration.
3. Grow start tree towards goal tree. At each iteration, select a random node in the goal tree to grow towards it.
4. Grow goal tree towards start tree. A solution path is found when the two trees finally connect.

We use the implementation of the RRT planner from the open-source library OpenRAVE [6].

4.2.3 Constrained Planning

For moving the grasped object in the environment, it is essential to maintain the relative configurations of the two end-effectors after the object has been grasped. Thus the object always stays fixed within the grasp while it is being moved. This is passed as a qualification constraint for every new sample configuration generated in the RRT-Connect planner and in the path smoothing process [10]. This state-qualification algorithm is shown in Algorithm 3. The initial relative configuration between the two end-effectors ($T\_relative$) is determined in the beginning. During the sampling phase, a new 11 DOF sample (S) is generated for the 4 joints of the torso and 7 joints of one of the arms (called primary arm). The torso and the primary arm are set to the sampled joint angles and the end-effector transform for this arm ($T\_EEF_1\_new$) is determined using the robot Forward Kinematics. Then, the end-effector configuration of the second arm ($T\_EEF_2\_new$) is determined such that the relative configuration between the two end-effectors corresponds to ($T\_relative$). A fast closed form inverse kinematics solver (IKFast [6]) is then used to calculate the joint positions for the 7 joints of the second arm($State_{Manip_2}$). This is similar to the fixed grasp dual arm path planner discussed in [21]. If an inverse kinematics solution exists, it is checked for lying in the vicinty of the second arm's old state. This is necessary since multiple inverse kinematics solutions can exist for a redundant manipulator and it has to be ensured that an inverse kinematics solution lying too far away from the initial state is rejected. Thereafter, the inverse kinematics solution is appended to state (S), which now contains information for all the robot joints. Finally, the robot is checked for collisions with the environment, and if it is collision-free, the final state is returned. If any of the constraint is not satisfied, the algorithm returns an empty state.

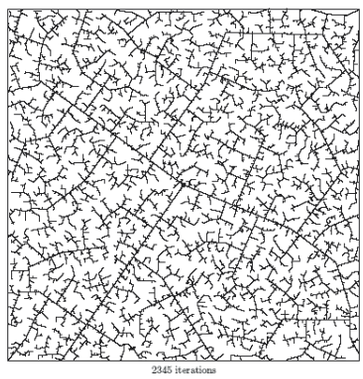


Figure 7: A dense RRT graph, source: http://en.wikipedia.org/wiki/Rapidly-exploring_Random_Tree, accessed 2012/11/25

Algorithm 3 State Qualification for Dual Arm Constrained Motion Planning

Determine Initial Relative Grasp Configuration between the two EEFs:
  $T\_EEF_1\_org = Manip_1 \rightarrow GetEEFTransform()$
  $T\_EEF_2\_org = Manip_2 \rightarrow GetEEFTransform()$
  $T\_relative = T\_EEF_1\_org.Inverse() * T\_EEF_2\_org$
for all new states S **do**
  $robot \rightarrow SetConfiguration(S)$
  $T\_EEF_1\_new = Manip_1 \rightarrow GetEEFTransform()$
  $T\_EEF_2\_new = T\_EEF_1\_new * T\_relative$
  $State_{Manip_2} = Manip_2 \rightarrow FindIK(T\_EEF_2\_new)$
  if ($State_{Manip_2}$ *is not none*) **then**
    if ($Difference(State\_org_{Manip_2}, State_{Manip_2}) \leq max\_threshold$) **then**
      $S = COMBINE(State_{Manip_1}, State_{Manip_2})$
      $robot \rightarrow SetConfiguration(S)$
      if ($!Environment \rightarrow CheckCollisions()$) **then**
        return S
      end if
    end if
  end if
  return Null
end for

Our dual-arm constrained planner as well as the dual-arm IK solver are available as the *dualarmmanipulation* plugin in OpenRAVE [6].

4.2.4 Trajectory Generation

After the joint level path planning, the maximum limits of each joint velocity are used to assign timestamps to each intermediate point. The time between two consecutive points is determined by the communication rate for the robot controller.

4.3 Motion Planning Architecture for fault management

For real world applications with the robot, the motion planning modules have to be interfaced, amongst others, with the robot vision. For the entire architecture to work reliably, a fault handling motion planning architecture has been developed for maintaining a world model and generating motion plans. The components of this architecture are discussed below.

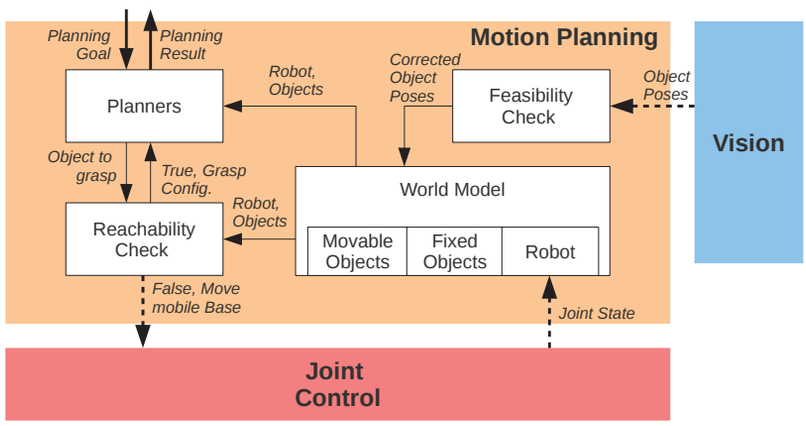


Figure 8: Combining low level motion planning architecture scripts for a grasp object application. Dashed lines represent ROS topics, continous lines internal data flow and bold continuous lines ROS action goals/results

4.3.1 Maintaining a World Model

An environment model is maintained that keeps track of the positions of the movable and stationary objects and the robot. The positions of objects are updated with inputs from the vision system. The robot's position changes are determined by self-localization w.r.t. a fixed object like a shelf or a table or via absolute displacement of the mobile base as calculated by the navigation controller [11].

4.3.2 Fault detection and error handling

Features like validating object pose feasibility and checking object grasp reachability are necessary for detecting and handling faults, and transitioning the system to the appropriate state.

Validating object pose feasibility This assumes that the ground surface is planar and the robot and fixed objects like tables and shelves are always placed flat on the ground plane. If the poses passed by the vision system adhere to this assumption within a threshold, the poses are corrected to make the objects parallel to the ground. Similarly, the movable objects are assumed to lie on a table or shelf and are corrected to lie on a plane parallel to the ground. In case the poses lie outside the threshold, a call is returned to the vision system to detect the object pose again.

Checking object grasp reachability Before a motion planner is invoked to plan the grasping motion, the object reachability is checked using the IK solvers. If the object is reachable, the grasping configuration is passed as the goal configuration to the planner. In case the object is not reachable, the robot base is commanded to move in front of the object from where the probability of reaching the object is maximal.

The motion planning architecture provides scripts which are integrated into the task level architecture using ROS action servers. The interaction of the components used for motion planning is shown in Figure 8.

5 Sensor Based Control

Sensor based control, including modalities like force sensors, stereo cameras and tactile sensors is a prerequisite for any robotic system that operates in uncertain environments. The input from external sensors can either fully guide the robot to its target or correct pre-planned trajectories in a reactive manner in order to overcome uncertainties or unexpected changes in the environment. The most commonly used sensors in robotics are vision and force sensors. AILA is equipped with a stereo camera system mounted in the head of the robot and two 6 DOF force/torque sensors mounted on each of the forearms. Up to now, two sensor based control techniques have been implemented and tested with the system, namely force control and Visual Servoing.

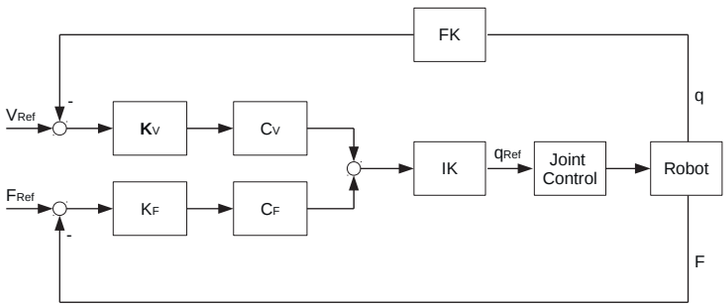


Figure 9: Parallel Force/Velocity Control scheme of AILA

5.1 Force Control

The inverse kinematics solvers described in section 3 can also be applied in Cartesian force control, using the sensor data from the 6 DOF force/torque sensor on the forearm of AILA. Therefore, a parallel force/velocity control scheme has been implemented as shown in figure 9. Like this, the robot can react to forces that are exerted on the end effector. Thus it achieves compliant behavior, which prevents damaging the robot, when hitting an obstacle unexpectedly. The control law is computed as follows:

$$\dot{\mathbf{q}}_{ref} = \mathbf{J}^*(\mathbf{q})\mathbf{K}_F\mathbf{C}_F(\mathbf{F}_{ref} - \mathbf{F}) + \mathbf{K}_V\mathbf{C}_V(\mathbf{V}_{ref} - \mathbf{J}(\mathbf{q}) \cdot \dot{\mathbf{q}}), \quad \mathbf{C}_V \perp \mathbf{C}_F \tag{12}$$

Here, $\mathbf{J}^*$ is the damped least squares inverse of the Jacobian $\mathbf{J}$, as described in section 3 and $\mathbf{K_F}$ and $\mathbf{K_V}$ the vectors of force and velocity control gains, respectively. $\mathbf{F}$ and $\mathbf{F}_{ref}$ are the actual and desired Cartesian wrenches and $\mathbf{V} = \mathbf{J}(\mathbf{q}) \cdot \dot{\mathbf{q}}$ and $\mathbf{V}_{ref}$ the actual and desired Cartesian velocities, respectively. The matrices $\mathbf{C}_F$ and $\mathbf{C}_V$ are called selection matrices. They decide which directions shall be force controlled and which directions shall be velocity controlled. Here, the matrices are chosen orthogonal to each other so that $\mathbf{C}_V\mathbf{C}_F = \mathbf{I}$. This decouples the force and velocity controlled directions and helps avoiding stability issues. However, in an uncertain environment or in complex interaction scenarios including e.g. non-rigid contacts, the distinction between force and velocity controlled directions may be unclear, dynamically changing or even impossible. Thus, a more sophisticated control strategy is required, allowing the description of more complex interaction tasks, like e.g. in [18].

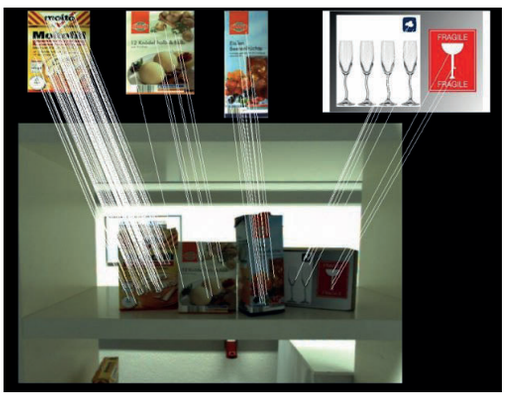
5.2 Visual Servoing

Visual Servoing (VS) has been introduced by Hill and Park [7] and is a commonly used sensor based control method in robotics. In VS, the motion of a robotic manipulator is controlled by the use of visual feedback. Generally, VS techniques can be classified into image based approaches (IBVS), where a set of image features is used to control the manipulator motion, position based approaches (PBVS), where the full pose of an object is used to guide the robot to its target position and hybrid methods that are a combination of both. Furthermore, different camera set-ups exist, usually classified as eye-in-hand, where the camera is mounted rigidly with respect to the robot's end effector, and eye-to-hand, where the camera is mounted fixed with respect to the world.

Regarding AILA, imprecise eye-hand-calibration, which arises from inaccurate joint zero position settings and imprecise link length measurements, leads to deviations between estimated and actual end effector pose up to a couple of centimeters. Visual Servoing is a method to overcome these inaccuracies. In the following sections, the methods used for object recognition and tracking, which lay the basis for the VS approaches used in AILA are described. Finally, the VS algorithm used is explained.

Object Recognition Object recognition is used in order to identify and localize manipulable objects within the operating area of the robot. We make use of textured 3D models, which store global object properties like size and shape, as well as local, texture based features. Object models are generated off-line, using images of the objects of interest. For feature extraction we use a GPU based implementation of SIFT[6]. The use of a graphics processing unit assures much faster computation times as compared to standard SIFT implementations, while maintaining its robustness against rotation, scale and change in illumination. The

[6]http://cs.unc.edu/ ccwu/siftgpu, accessed 2013/03/18



(a) Feature matching: The extracted features are matched with textured 3D models, which are kept in a model database

(b) 3D pose estimation: Using a homographic mapping between model and camera image plane, the pose of the objects in 3D space can be estimated.

Figure 10: Recognition of textured object using SIFT

extracted features are matched with the previously mentioned model database, using the modified k-d tree algorithm described in [12]. For efficiently clustering model hypotheses, we use the RANSAC algorithm. It rejects outliers and helps computing a homographic mapping from the model into the camera image plane, which, again, can be used to find certain boundary points of the object within the camera image. These points and their correspondences in the 3D model of the object are used as input for pose estimation, where we make use of a simple iterative algorithm for pose refinement in addition to a homography based method to get an initial pose estimate. The result of matching and pose estimation of textured objects can be seen in Figure 5.2.

In case an object is far from the camera, partly occluded or the image data is noisy, the computed object poses may be incorrect and subsequent grasping is likely to fail. In order to reduce the risk of executing a grasp on faulty object data, the score value s_m is introduced at matching level, which shall be described as follows:

In the matching approach described in [12] feature pairs are only accepted as a match, if the euclidean distance in feature space to the closest match, d_0, is much less then the distance to the second closest match d_1, or

$$d_0 < d_1 \cdot T_{NN}, \quad 0 < T_{NN} < 1 \tag{13}$$

The nearest neighbor threshold T_{NN} determines the amount of discarded features. If T_{NN} is too high, many false positives will occur, if it is too low, many correct matches will be discarded. Now, the score value s_m is computed as the sum of distances of the second closest neighbors divided by the sum of distances of the closest neighbors for all matches:

$$s_m = \frac{1}{K_N} \cdot \sum_{i=1}^{i=N_M} \frac{d_{1,i}+1}{d_{0,i}+1} \quad \forall i,\ d_{0,i} < d_{1,i} \cdot T_{NN} \tag{14}$$

Here N_M is the number of matches and K_N is a normalization factor, that takes into account all features, also the discarded ones:

$$K_N = \sum_{i=1}^{i=N_M} \frac{d_{1,i}+1}{d_{0,i}+1} \quad \forall i \tag{15}$$

The value of s_m increases if many matches with $d_0 << d_1$ occur, which is an indicator of a hypothesis being correct. An object hypothesis is discarded if

$$s_m < T_M, \quad 0 < T_M < 1 \tag{16}$$

where T_M is the matching threshold. The value of this threshold was determined empirically to $T_M = 0.4$.

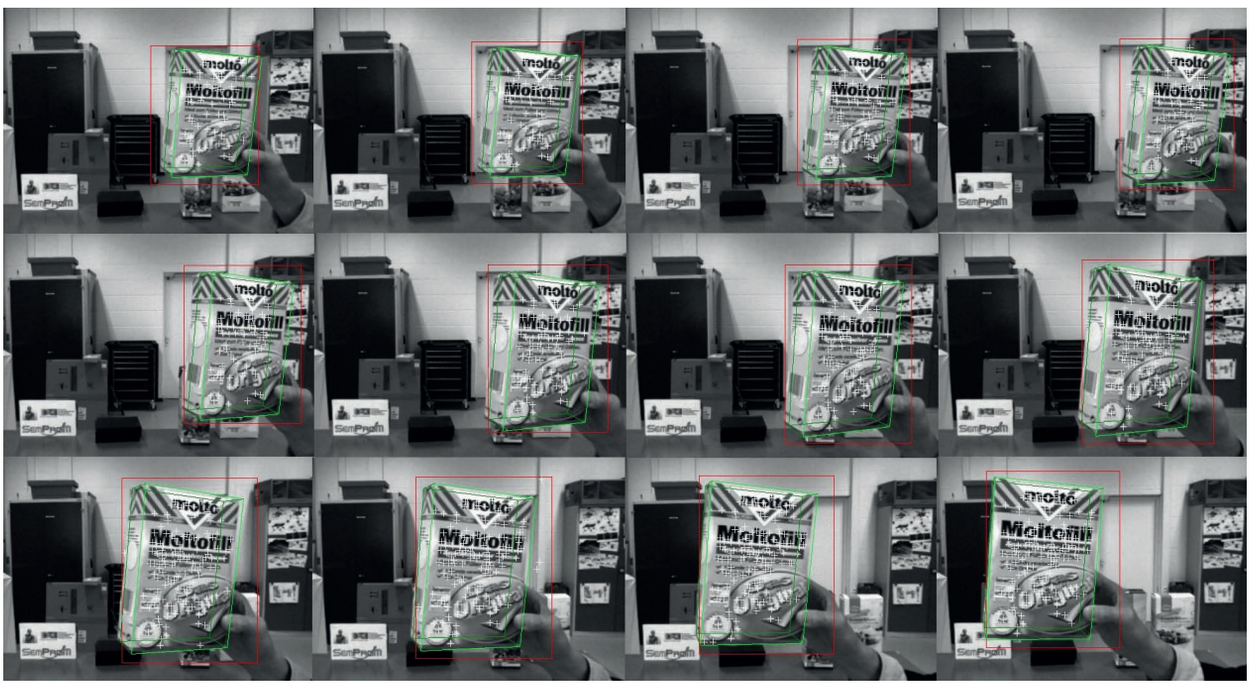


Figure 11: Tracking sequence: The red box denotes the current region of interest, the white crosses are the tracked SIFT features and the green wire frame denotes the estimated object pose.

Like this, only objects with a high probability of being correctly recognized[7] are published via the ROS network.

Object Tracking Object tracking as shown in Figure 11 is required for the Visual Servoing in order to continuously estimate the pose of the object of interest. For tracking, we make use of a Kalman filter in combination with the GPU based implementation of SIFT as mentioned earlier. The object's pose, acquired from the previously described object recognition, serves as input for the Kalman Filter which estimates the object pose in the successive image frame. By back projection of the object boundaries into the image plane, a region of interest for the next feature extraction step is generated. Like this, only a small portion of the image is used for the computationally expensive extraction of the SIFT features. By the means of this approach, frame rates between 25 and 50 Hz [8] can be obtained on a dual-core 1.8 GHz processor and a NVidia GT440 graphics card. In case the object pose is lost, the ROI is augmented exponentially, until the object was found again, or the ROI size exceeds the camera image size, in which case the tracking is terminated.

VS Algorithm Due to the fact that a suitable object pose estimation algorithm is already available (see above) a position based VS has been implemented. Since AILA is not equipped with a wrist mounted camera, an eye-to-hand VS scheme has to be applied, which means that both, the end effector and the object of interest must be tracked simultaneously. In order to robustly track the end effector, a colored marker as shown in figure 12, is being used. The difference in position (and orientation) between both can be used to generate a velocity signal, which drives the end effector to the target:

$$\dot{\mathbf{q}}_{ref} = \mathbf{J}^*(\mathbf{q}) \cdot (K_p(\hat{\mathbf{X}}_O(t) - \hat{\mathbf{X}}_M(t)) + K_I \sum_{\tau=t_0}^{t} \hat{\mathbf{X}}_O(\tau) - \hat{\mathbf{X}}_M(\tau)) \tag{17}$$

Here, $\mathbf{J}^*(\mathbf{q})$ is the weighted pseudo inverse of the robot Jacobian as described in section 3, $\dot{\mathbf{q}}_{ref}$ the

[7] Note that a high score value means that the object pose is correct with a high probability. A low score value on the other hand does not necessarily mean that the computed pose is wrong. However, we accept the risk of discarding some correctly recognized objects in the field of view, since hypotheses with a low confidence can be further examined by changing the camera view.

[8] Depending on the size of the object of interest and the amount of texture in the camera image

(a) Color Blob Detection and Regions of Interest

(b) Marker Pose Estimation (xyz-axis represented in red, green and blue color, respectively) and Control panel tracking using SIFT image features

Figure 12: Hand Pose Tracking using a colored marker

reference joint velocity, $\hat{\mathbf{X}}_O(t)$ the estimated object pose and $\hat{\mathbf{X}}_M(t)$ the estimated marker pose. K_p and K_I are the proportional and integral gain, respectively, which have been determined experimentally.

The marker is recognized in the camera image using HSV image based color blob detection and tracked with a Kalman Filter approach. Thereby, the estimated pose given by the (inaccurate) forward kinematics of the robot can be used as initial guess. The pose of the marker is computed using the plane connecting the three differently colored blobs. The object of interest (in this case the control panel shown in the figure) is first recognized and then tracked using the methods described in the previous two sections.

6 Telemanipulation

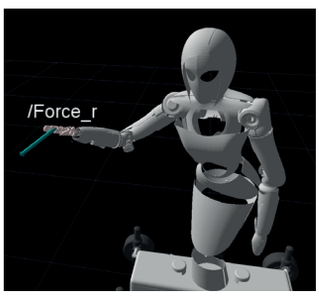
In robotic telemanipulation you can distinguish end effector based and joint-to-joint telemanipulation. Using the former, the operator controls the robotic end effector in Cartesian space, so that relatively simple input devices are sufficient. Furthermore, the control scheme will be applicable to a large number of robotic systems, since it is independent of the morphology of the robot. However, using end effector based control the robot will be more difficult to operate, as compared to joint-to-joint control, which provides a very natural feeling of telepresence to the human operator. With such a scheme a more sophisticated, whole arm interface and a joint-to-joint mapping is required, which might be difficult to obtain, depending on the morphology of the robot and the operator interface. In the following sections, we describe our approaches on end effector based and joint-to-joint teleoperation with AILA.

6.1 End Effector Based Telemanipulation

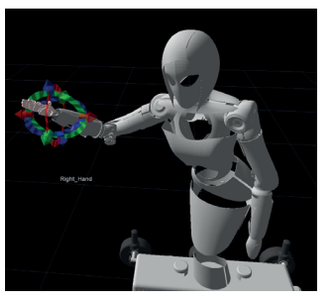
For end effector based teleoperation we use the Space Navigator 6 DOF mouse[9] It allows translational and rotational motion around three axes and can therefore be used to teleoperate AILA in Cartesian space. In order to communicate with the robot, the signals from the Space Navigator are scaled to velocity signals, transformed to the robot base frame and then sent to the Cartesian Control component (see figure 2), which performs inverse kinematics and communicates with the joint control interface. From the software side, the teleoperation happens in open-loop mode. However, the operator receives feedback from the visual sensors of AILA and has to adapt his behavior accordingly (*operator in the loop*). In order to get contact information, the sensor data from the force sensors is displayed in the visual model of AILA (See figure 13(a)).

Another possibility is to use the interactive marker functionality of the ROS visualization tool Rviz (see figure 13(b)). Here, a six DOF marker is displayed in the visualized model of AILA and the end effectors can be simply dragged with a normal desktop mouse. The underlying control mechanisms are the same.

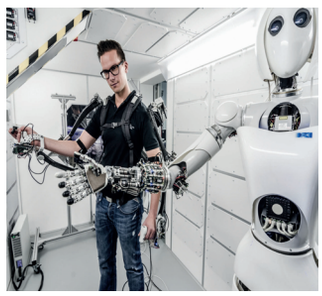
[9] http://www.3dconnexion.com/products/spacenavigator.html, accessed: 2013/03/18



(a) Semi-Haptic Feedback: Visualization of the forces exerted on the end effector



(b) End effector control using interactive markers in rviz



(c) Joint-to-joint telemanipulation using dual-arm, passive Exoskeleton

Figure 13: Teleoperation with AILA

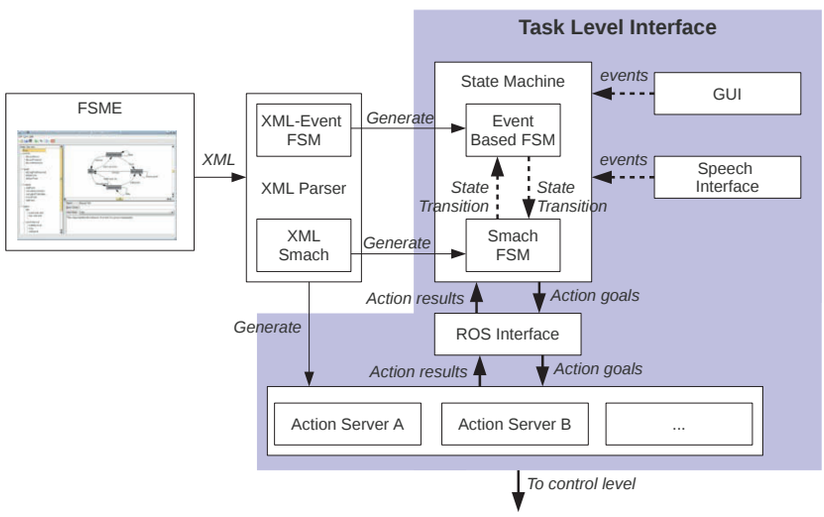


Figure 14: Finite State Machine Editor (FSME), XML-Parsing and integration in ROS. Dashed lines represent ROS topics, continuous lines internal data flow and bold continuous lines ROS action goals/results

6.2 Whole Arm Telemanipulation

Within the Capio project of DFKI[10], a passive, dual-arm exoskeleton has been developed (see figure 13(c)), which can measure joint displacements of both arms and the upper body of the human operator. The displacements are sent as joint position commands to AILA via ROS. Due to the human-like morphology of the exoskeleton, it is possible to obtain reasonable results with a 1:1 joint-to-joint mapping. However, in order to improve accuracy, a more complex mapping will be implemented in the future. The hands of AILA can be controlled using the buttons at the hand interface of the exoskeleton. Like this, it is possible to use predefined grasp configurations of AILA's hands, in order to enable the operator to pick up known objects. An inertia measurement unit, which is mounted on the head of the human operator is used to control the pan-tilt unit in the head of the robot. Additional improvements, like force and tactile feedback are planned for the future.

7 Task Level Interface

In the previous sections, motion control methods in Cartesian and joint space have been described. Now, these controllers have to be coordinated and configured appropriately considering the task that the robot has to accomplish. This coordination should take place on a higher, abstract level, commonly referred to as task level.

[10] http://robotik.dfki-bremen.de/en/research/projects/capio-1.html, accessed 2012/12/7

For task coordination, we use the Python architecture Smach [3], which can be used for quickly prototyping and introspecting state machines and nicely integrates with ROS high level concepts. However, due to its rather complicated python syntax, larger state machines are difficult to read and debug, while the code becomes error prone. Another problem is that Smach does not create event based state machines in the classical sense, but rather a script of tasks that it should execute in a certain order, depending on the tasks output. For applications like voice control, where the robot remains in a specific state for a long time, and changes the state when a voice command comes in, Smach is not well suited.

Therefore, we use the GUI based Finite State Machine Editor[11]. Here, it is possible to easily develop event based state machines by dragging and dropping the states on the graphical surface. FSME stores its state machines in an XML-format. It comes with a parser that can convert those files into a python script. In order to comply with the ROS high level concepts, another parser has been implemented, which is able to create Smach state machines from FSME files. Furthermore, the bridge between the event based FSME state machines and Smach has been implemented. The parsing process and the integration with the ROS framework is illustrated in figure 14. The concept of automatic code generation, which is used here, speeds up the development process and produces less error prone, consistent code. Each state that is generated in FSME automatically creates a state in the Smach state machine and a ROS action server, which can either be filled with code by the user or point to an already existing action. The action servers connect via the ROS interface to the state machine and to the lower level parts of the architecture (see figure 2).

Events can be generated either from a GUI or via a voice control interface. For voice recognition we use CMUSphinx[12], which is a leading speech recognition toolkit with various tools used to build speech applications. The speech input triggers events, which are communicated via ROS and activate certain behaviors of the robot. Furthermore, the robot is able to reply with a couple of predefined sentences using speech generation. As a tool for speech generation, we use the Festival speech synthesis system which has been developed by the university of Edinburgh [20].

8 Experimental Results

8.1 Vision

We evaluated the SIFT GPU based object recognition by the means of more than 400 test images, each showing one or more of 10 test objects in different poses. The performance is described by the object pose accuracy D:

$$D = e^{-\frac{1}{C}\sum_{i=1}^{N}\Delta d_i} \qquad 0 \leq D \leq 1 \tag{18}$$

where N is the number of object boundary points, C is a normalizing factor and Δd_i is the Euclidean distance in 3D space between the estimated position of the i-th boundary point and their real position (ground truth). The results are displayed in Figure 15.

The algorithm is robust with respect to orientation and scale, which is related to the characteristics of SIFT. All in all, objects can be correctly recognized up to a rotation of 40° with respect to the camera image plane and a scale[13] of 2.5. The figures also offer clues about the relation of the score value s_m (see section 5) and the object pose accuracy. When $s_m > 0.4$ the object pose accuracy is high in both cases, so the object is recognized correctly with a high probability. Thus, the matching score threshold $T_M = 0.4$ can be used to discard or accept object hypotheses according to their reliability.

8.2 System Integration

As already mentioned in section 2, we evaluated two fundamentally different approaches for task execution. The first one is based on motion planning in static environments, the second is a reactive approach, using sensor based control.

We evaluated the functionality of our planning based approach by the means of a dual arm pick and place scenario in a logistics environment. Thereby, the robot's task is to autonomously grasp an object from a table with two hands and release it on a different table in the same room. Snapshots of the grasping sequence can be seen in Figure 16. Furthermore, a video showing the results can be viewed at YouTube[14].

[11] http://fsme.sourceforge.net, accessed 2012/12/7
[12] http://cmusphinx.sourceforge.net/wiki. Accessed: 26/06/2012
[13] Ratio of model image size to object size in the camera image
[14] http://www. youtube.com/watch?v=xZ39K9JOayw, accessed: 28/01/2013

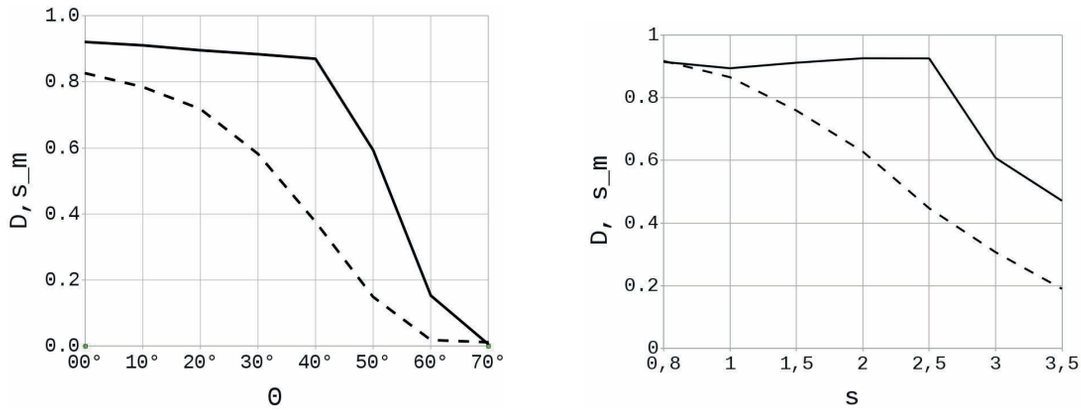


Figure 15: Results on the robustness of object recognition with respect to rotation (**left**) and scale (**right**). Continuous Line: Recognition accuracy D, Dashed line: Score value s_m.

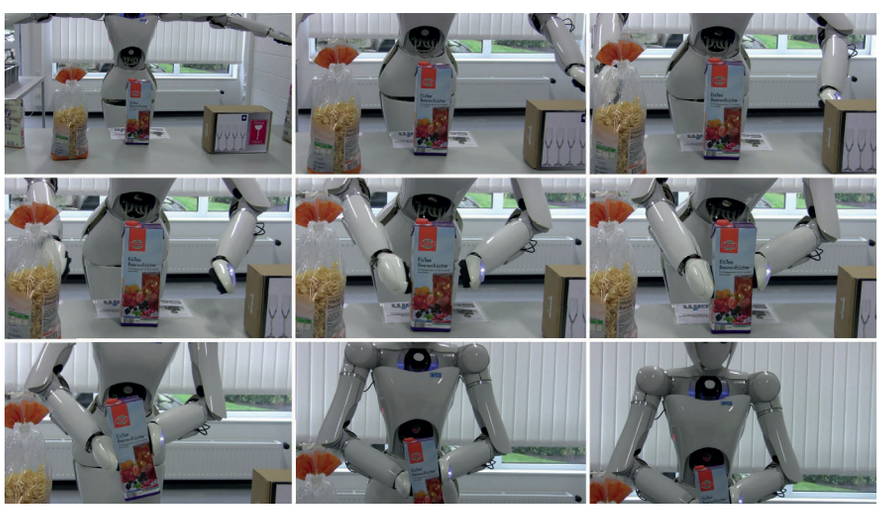


Figure 16: Screenshots of AILA grasping an object from a table using motion planning based dual arm manipulation

The robot recognizes the object using the vision system described in section 5, places it in the world model while checking the feasibility of the object's pose, computes an IK solution, plans a grasp and executes a dual arm trajectory. The final grasp is achieved and maintained with the help of the force sensors attached to the robot's arms. After grasping the object, the robot moves it from the table using the constrained planner described in section 4. The average computation times for the main subcomponents required for this process are shown in figure 17. Note that the trajectory execution is not included here, since it depends on the selected moving speed of the robot. For data collection, 50 experiments have been performed, where an object has been placed at different positions in an uncluttered environment within the reachable space of the robot. The measurements for the vision components have been performed on a mini-ITX Intel Dual Core CPU with 2.8GHz and a GeForce GT240 graphics card, the ones for the planning algorithms on a LS-372 motherboard populated with an Intel Dual Core 2.5GHz. The image size used for object recognition is 782x582, the model database contained 10 object models with 2 object views, respectively. It can be seen that the overall computation time, excluding the task execution, is around 8 seconds on average. Especially the constrained planner requires a high amount of computation, which is due to the difficulty of finding an IK solution and a motion plan in constrained space for both arms at the same time.

The sensor based approach has been demonstrated by means of a space scenario. Here, AILA has to perform simple manipulation tasks, like switching a button, in a mock-up of the International Space Station (ISS). These tasks can be achieved using Visual Servoing to guide the manipulator to the control panel and force control in order to observe the interaction with the environment. Figure 18 shows a sequence of AILA switching a button using such an approach. Furthermore, a video showing the results can be viewed at

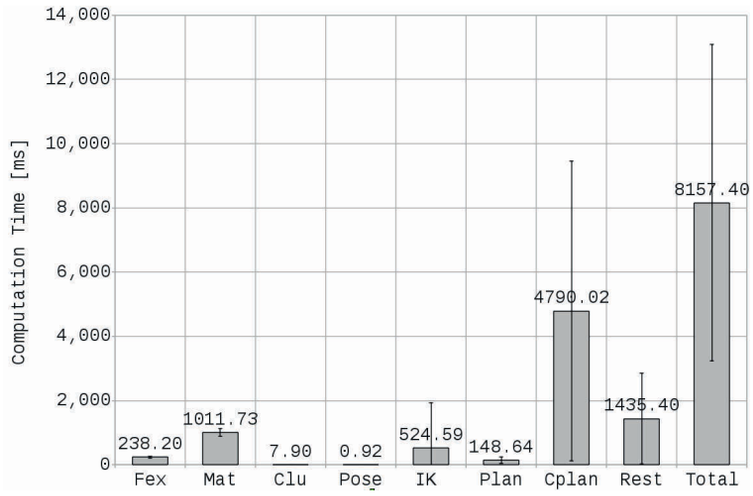


Figure 17: Average computation times in ms for the main components of dual arm grasping. Fex - Feature Extraction, Mat - Matching, Clu - Clustering, Pose - Pose Estimation, IK - Inverse Kinematics, Plan - Planning, Cplan - Constrained Planning, Rest - Other components and overhead

YouTube[15]. The task is performed using reactive control only, the robot can react on sensory events during execution. However, sensor based approaches may lead the system into local minimums. Especially when joint limits may be violated, the primary task might not be executed successfully and a global path planner has to resolve the situation.

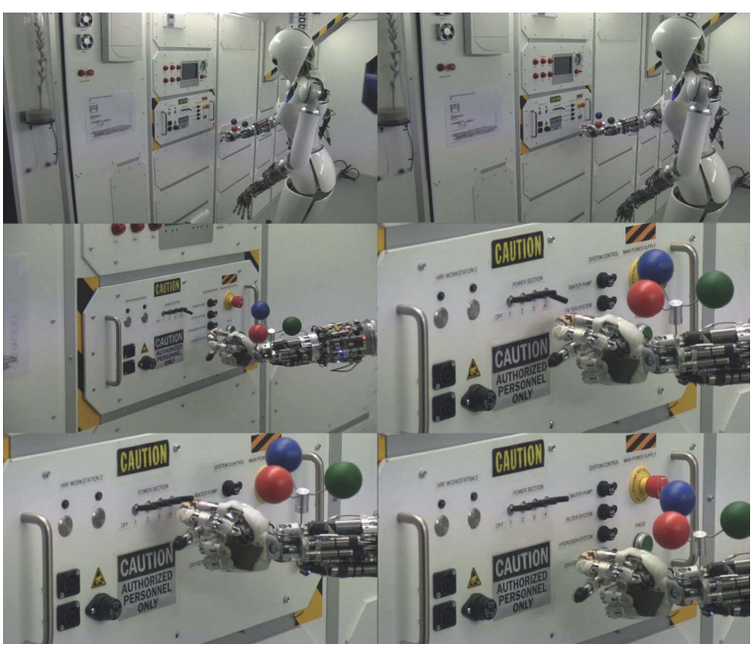


Figure 18: AILA is executing an autonomous, sensor based task in an ISS mock-up

9 Conclusion and Future Work

In this technical report we illustrated our efforts on the development of key software components for robot control, including motion control, planning, sensor based control, telemanipulation and task level control. These components represent the base requirements for the robotic system AILA to execute manipulation tasks. We presented results in form of experiments using a motion planning based approach in a logistics

[15]http: //www.youtube.com/watch?v=HTLgoS8AihM, accessed: 28/01/2013

scenario and a reactive, sensor based method in an ISS scenario. We showed that the robot is able to perform interaction tasks and object manipulation in uncluttered environments using previously known object models. However, in order to increase robustness and perform more complex interaction with an uncertain environment, further developments are necessary.

Many robotic systems, including AILA, feature a large number of degrees of freedom and multiple sensing modalities. Despite this rich sensory and kinematic set-up, the capabilities regarding sensor based control in object manipulation are rarely fully exploited. One focus of our future work will be on the development of sophisticated sensor based control approaches for redundant robots. Up to now, we developed force and vision based control strategies, which work independently of each other in a way that a visual sensor guides the robot to an object of interest, while force or tactile sensors help accomplishing the grasp. Such a strategy is bound to fail frequently in the presence of larger uncertainties (e.g. sensor noise, non-rigid contacts, etc.). The simultaneous use of feedback from multiple modalities is a way to improve robustness, speed and accuracy of such tasks. In particular, the following concepts of sensor based control shall be highlighted in future:

- *Task integration:* Different sensors can be used to control different subtasks simultaneously e.g. a vision sensor could control the Cartesian direction parallel to a given contact surface while a force sensor controls the direction perpendicular to the surface. In doing so, the redundancy of the robot can be exploited to comply with secondary tasks, such as joint limit avoidance or posture control.
- *Predictive Control:* Multiple sensing modalities can be used to predict required motor commands for interaction control. In particular, one sensor may be used to predict the required reference values for a control loop where a different sensor is employed.
- *Task Monitoring:* Discrete sensory events encoded in multiple modalities may indicate sub-goal attainment and thus lead to sophisticated task monitoring approaches by comparing predicted and actual sensory input. If a task cannot be solved in a reactive manner re-planning has to be triggered.
- *Sensor Fusion:* Multiple modalities can be fused to improve the amount of content or accuracy of the gained information, for example the shape of a grasped object.

Opposite to the static motion planning approach illustrated in section 4, where a *global* solution for the desired robot motion is found based on a world model, sensor based control is characterized by the ability to react *locally* on dynamic changes in the environment. However, reactive control approaches are prone to guide the robot to local minima, so that a global approach needs to resolve the situation. Thus, a sophisticated integration of planned and reactive behavior must be attained in order to successfully deal with unforeseen situations.

References

[1] T. Asfour, K. Regenstein, P. Azad, J. Schröder, N. Vahrenkamp, and R. Dillmann. Armar-iii: An integrated humanoid platform for sensory-motor control. In *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, pages 169–175, 2006.

[2] Sebastian Bartsch, Timo Birnschein, Malte Römmermann, Jens Hilljegerdes, Daniel Kühn, and Frank Kirchner. Development of the six-legged walking and climbing robot spaceclimber. *J. Field Robot.*, 29(3):506–532, May 2012.

[3] J. Bohren and S. Cousins. The smach high-level executive [ros news]. *Robotics Automation Magazine, IEEE*, 17(4):18–20, dec. 2010.

[4] Christoph Borst, Thomas Wimbock, Florian Schmidt, Matthias Fuchs, Bernhard Brunner, Franziska Zacharias, Paolo Robuffo Giordano, Rainer Konietschke, Wolfgang Sepp, Stefan Fuchs, Christian Rink, Alin Albu-Schaffer, and Gerd Hirzinger. Rollin' justin - mobile platform with variable base. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 1597–1598, may 2009.

[5] M.S. Branicky, M.M. Curtiss, J. Levine, and S. Morgan. Sampling-based planning and control. In *Proceedings of the 12th Yale Workshop on Adaptive and Learning Systems, New Haven, CT*. Citeseer, 2003.

[6] Rosen Diankov and James Kuffner. OpenRAVE: A planning architecture for autonomous robotics. Technical Report CMU-RI-TR-08-34, Robotics Institute, July 2008.

[7] J. Hill and W. Park. Real-time control of a robot with a mobile camera. In *In Proceedings of the 9th International Symposium on Industrial Robots*, pages 233–246, 1979.

[8] K. Kaneko, F. Kanehiro, S. Kajita, H. Hirukawa, T. Kawasaki, M. Hirata, K. Akachi, and T. Isozumi. Humanoid robot hrp-2. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 2, pages 1083–1090 Vol.2, 26-may 1, 2004.

[9] S. Komada, M. Ishida, K. Ohnishi, and T. Hori. Motion control of linear synchronous motors based on disturbance observer. In *Industrial Electronics Society, IECON â€™90, 16th Annual Conference of IEEE*, volume 1, pages 154–159, 1990,.

[10] J.J. Kuffner and S.M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pages 995–1001, April 2000.

[11] Johannes Lemburg, José de Gea Fernández, Markus Eich, Dennis Mronga, Peter Kampmann, Andreas Vogt, Achint Aggarwal, Yuping Shi, and Frank Kirchner. Aila - design of an autonomous mobile dual-arm robot. In *Proceedings of IEEE International Conference on Robotics and Automation*, 2011.

[12] David G. Lowe. Object recognition from local scale-invariant fatures. In *The Proceedings of the Seventh International Conference on Computer Vision*, volume 2, pages 1150–1157, 1999.

[13] J. Lunze. *Regelungstechnik 1.* Springer Verlag Berlin Heidelberg, 2012.

[14] K. Ohnishi, S. Masaaki, and T. Murakami. Motion control for advanced mechatronics. In *IEEE/ASME Transactions on Mechatronics*, volume 1, 1996.

[15] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. ROS: an open-source robot operating system. In *In Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[16] Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki, and K. Fujimura. The intelligent asimo: system overview and integration. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, pages 2478–2483 vol.3, 2002.

[17] B. Siciliano et al. *Robotics - Modelling, Planning and Control*, chapter 3, pages 124–127. Springer London, 2010.

[18] R. Smits, T. De Laet, K. Claes, H. Bruyninckx, and J. De Schutter. itasc: a tool for multi-sensor integration in robot manipulation. In *Multisensor Fusion and Integration for Intelligent Systems, 2008. MFI 2008. IEEE International Conference on*, pages 426–433, aug. 2008.

[19] H. Täubig, , U. Freese, and B. Bäuml. Real-time swept volume and distance computation for self collision detection. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011.

[20] Paul A Taylor, Alan Black, and Richard Caley. The architecture of the festival speech synthesis system. In *The Third ESCA Workshop in Speech Synthesis*, pages 147–151, Jenolan Caves, Australia, 1998.

[21] N. Vahrenkamp, M. Do, T. Asfour, and R. Dillmann. Integrated grasp and motion planning. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2883–2888, may 2010.

German Research Center for Artificial Intelligence (DFKI) GmbH

DFKI Bremen
Robert-Hooke-Straße 5
28359 Bremen
Germany
Phone: +49 421 178 45 4100
Fax: +49 421 178 45 4150

DFKI Saarbrücken
Stuhlsatzenhausweg 3
Campus D3 2
66123 Saarbrücken
Germany
Phone: +49 681 875 75 0
Fax: +49 681 857 75 5341

DFKI Kaiserslautern
Trippstadter Straße 122
67608 Kaiserslautern
Germany
Phone: +49 631 205 75 0
Fax: +49 631 205 75 5030

DFKI Projektbüro Berlin
Alt-Moabit 91c
10559 Berlin
Germany
Phone: +49 30 238 95 1800

E-mail:
reports@dfki.de

Further information:
http://www.dfki.de