Planning and Execution of Grasping Motions on a Humanoid Robot

Nikolaus Vahrenkamp, Anatoli Barski, Tamim Asfour and Rüdiger Dillmann
Institute for Anthropomatics
University of Karlsruhe
Haid-und-Neu-Strasse 7, 76131 Karlsruhe, Germany
Email: {vahrenkamp,barski,asfour,dillmann}@ira.uka.de

***Abstract*— In this paper we present an approach for generating collision-free grasping motions and robustly execute them on a humanoid robot. The proposed *MultiEEF-RRT* algorithm for planning collision-free grasping trajectories exploits the enlarged goal space of a humanoid robot that results from the parallelized search of grasping trajectories for each arm. Here, multiple paths are searched simultaneously and the planner automatically chooses the first found solution. The reactive execution component operates on the planned C-Space trajectories and observes the movements in workspace with visual servoing approaches. The proposed algorithms do not rely on hand-eye calibrations, however it is possible to reliably execute given trajectories. The approach is fault-tolerant against changing execution speed, inaccurate sensor data and inexact executions of velocities. Since the hand and the target poses are visually tracked, the Cartesian error between the estimated position on a trajectory and the visually retrieved hand pose can be determined in workspace. This value is projected in the configuration space and used as a correction factor when calculating the joint velocities. We realized a grasping scenario with the humanoid robot ARMAR-III, where an object in front of the robot should be grasped. This demonstration shows how the proposed components play together to build a reactive and robust system integrating planning and execution of collision-free motions.**

I. INTRODUCTION

Humanoid robots are developed to work in human-centered environments and to assist people in doing the housework. To enable the robot operating in a safe and robust manner, a lot of components have to collude and to operate cooperatively. In this work we want to show how manipulation tasks can be planned by RRT-based planners and executed based on position-based visual servoing (PBVS). Here, the focus lies on the execution of planned trajectories and how visual information can be used to increase the accuracy. The proposed algorithms are implemented on the humanoid robot ARMAR-III [1] and evaluated with different test setups.

A typical manipulation task, like grasping an object requires several components as depicted in Fig. 1. In section III the perceptional component is briefly discussed. Since the computer vision approaches are not scope of this work, we just give a short overview of the used algorithms and how the internal representation of the scene is built. In section IV the planning framework is presented and the a parallelized RRT-based planner which considers multiple end effectors of a robot is introduced. The proposed *MultiEEF-RRT* planner can handle multiple end effectors (EEF) for single-handed grasping of an object and implicitly selects a feasible EEF and a suitable grasp for which a collision-free grasping trajectory exists. The planned trajectories are executed using visual servoing approaches as described in section V.

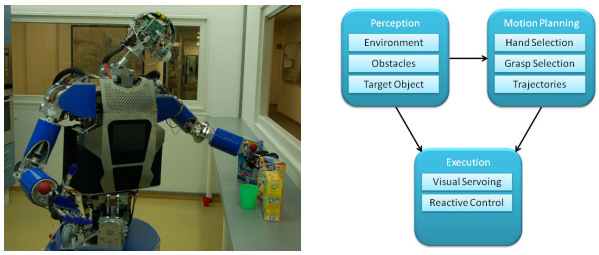

Fig. 1. (a) The Humanoid Robot ARMAR-III grasping an object. (b) The Components *Perception*, *Motion Planning* and *Execution*.

II. RELATED WORK

In [2] a real-time system for collision avoidance on a humanoid robot is presented which implicitly avoids obstacles during reaching a specific goal in workspace. The work of Berenson et. al. [3] deals with selecting feasible grasps out of a predefined set in cluttered environments for planning collision-free grasping motions for a humanoid robot. The vision-based approaches in [4] or [5] can be used to generate collision-free trajectories for a humanoid robot in unknown environments. Gienger et. al. present an approach for solving the coupled problem of reaching and grasping an object in a cluttered environment with a humanoid robot by using task maps which represent the manifold of feasible grasps for an object [6]. The capability maps, introduced in [7], can be used to analyze the reachability of objects with dual-arm robot systems. In [8] Paulin presented an approach where image-based visual servoing (IBVS) techniques are used for controlling the trajectory execution of a *camera-in-hand* robot arm system. Here the feature trajectory in the image plane is used for generating control commands. The visual servoing approach presented in [9] avoids pre-planned trajectories by implicitly avoiding obstacles during mobile navigation.

III. PERCEPTION

Usually the robot's framework provides an internal representation of the robot and the world, e.g. a virtual environment with 3D models of the surrounding, the manipulation

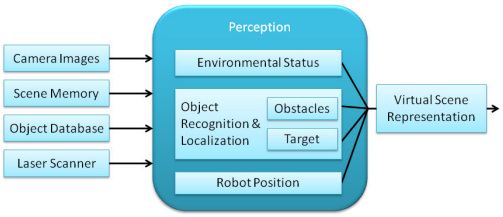

Fig. 2. Building a virtual representation from perceptual data.

objects and the robot. These models can be used for collision detection or distance calculation and thus as an input for the motion planning algorithms. To build up the internal representation, the software framework of ARMAR-III offers a predefined scenegraph of the rooms in which the robot operates. This basic information is updated according to the environmental status that is determined by sensor information, e.g. the angles of open doors can be visually detected. Additionally to the mostly fixed environment, the robot must be able to recognize and localize manipulation objects and humans in order to build up a valid representation of the surrounding. For planning motions the exact localization of obstacles and manipulation targets is crucial and the utilized algorithms for computer vision are able to supply highly accurate 6D object poses with high frame rates [10]. Furthermore the position of the robot has to be registered in the virtual environment. In case of ARMAR-III three onboard laser scanners are used to continuously localize the robot (see [1]).

IV. RRT-Based Motion Planning

Motion planning algorithms are used to search collision-free trajectories that move the robot to a desired goal which can be defined in work space or in the configuration space (C-Space) of the robot. Since the general motion planning problem is P-SPACE hard [11], complete algorithms are time consuming or even not available for high dimensional problems. Probabilistic approaches have been developed which are used to find a solution quickly but never terminate if no solution exists [12]. Sampling-based planning algorithms, like the Rapidly Exploring Random Trees (RRT), use random samples to explore the collision-free configuration space (C_{free}) and through their probabilistic completeness it is guaranteed that a solution is found if at least one exists [13], [14].

A. The Planning Framework

Depending on the requested task and based on the internal representation of the surrounding, the component for planning motions computes a collision-free trajectory for the involved joints. Which joints of the robot are used for planning depends on the task and the current state of the robot. E.g. if the robot should grasp an object which is far away, the joints responsible for moving the robot around have to be included, otherwise, if the object is reachable by an arm, the position of the robot can remain unchanged and thus the dimension of the C-Space is lower.

To plan the motion different motion planning algorithms can be selected and parametrized. In this paper we want to focus on an extension of the IK-RRT approach which can be used for single and dual arm planning problems [15]. The IK-RRT algorithm does not require a C-Space target configuration like most RRT-based planners, but it operates on a set of feasible grasping positions, defined as transformations in workspace relative to an object, and via a probabilistic IK-solver potential C-Space targets are calculated during the planning process. By avoiding pre-calculated IK solutions, the advantage of redundant robot systems are utilized by the planner, resulting in a efficient RRT-based planning algorithm which combines the search for IK solutions with the search for collision-free grasping motions.

B. Considering Multiple End Effectors

In this work we investigate how trajectories for grasping can be planned by RRT-based motion planning algorithms. Depending on higher-level task planning components, the goal of grasping an object can result in different constraints for planning:

Exact goal configuration:
The requested planning task definition specifies the exact target configuration which can be derived by IK-solvers or through predefined setups of the robot. These target configurations can be used as input for a standard Bi-RRT algorithm [13].

Hand and grasp selection:
The planning is constrained by specifying the hand, the exact grasp type and all parameters, needed for grasping the object. This behavior can be useful if the manipulation actions that may follow the grasping action need the object grasped in a defined way. Here the target configuration can be defined in work space and the planner solves the IK-problem by its own.

Hand selection:
The specification of the planning task includes which hand and which object should be used for planning the grasping motion, but which grasp out of a set of possible grasps should be used is left open. This behavior may be useful when the second arm of the humanoid is already holding another object, and thus the selection which arm should be used for grasping must be specified in advance.

No selection of hand or grasp:
The only request is that the object should be grasped and the selection which hand and what type of grasp should be used is disposed to the motion planning framework. Here the goal is to grasp the object and e.g. to hand it over to a human operator. In this case the generation of target configurations is transfered from the task planning level to the motion planning algorithms.

Specifying less parameters in advance results in a larger goal space and thus the chance of finding a path to one configuration in C_{goal} increases. If the planning framework leaves it open if the left or the right hand should be used for grasping, the proposed approach can consider both kinematic

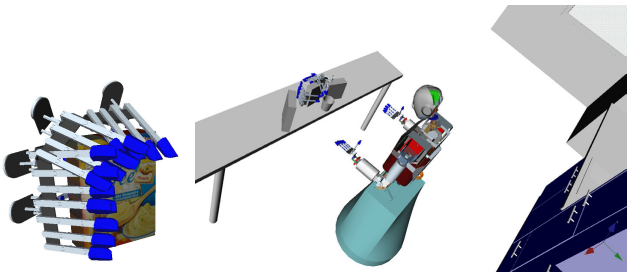

Fig. 3. (a) A 3D model of a mashed potatoes box with four associated grasps of the right hand. (b) A virtual planning scene where the position of the robot is retrieved by the internal laser scanners and the object positions are determined by the vision system.

chains with a predefined set of grasps for each hand and thus an extended goal space can be used.

C. MultiEEF-RRT

The proposed *MultiEEF-RRT* planner can be used in case the goal is to grasp an object and the planner has to decide which end effector (EEF) should be used to grasp that object. The planner is initialized with the state of the robot and the environment, the pose of the object which should be grasped and a set of feasible grasps for each EEF of the robot (in case of a humanoid robot the two hands are considered). The set of feasible grasps combined with the object position is used to calculate a set of possible target poses in workspace for each end effector. These sets of target poses are used during the planning process to calculate C-Space target configurations for both end effectors via a probabilistic IK-Solver. Since multiple EEFs, each with multiple possible grasping poses, should be considered and there is no knowledge about the reachibility of the grasps, the *MultiEEF-RRT* planner parallelizes the search for a collision-free grasping trajectory. A thread is started for each EEF and the planning with the kinematic chain responsible for moving this EEF (e.g. the arm) is done independently from the other EEFs. If the planning problem for one EEF could be solved by a sub-planning thread, the other threads are stopped and a global solution, containing the EEF and grasp selection, the corresponding kinematic chain and the trajectory, is reported. As shown in Fig. 4 the virtual representations of the scene and the robot have to be cloned n times (where n is the number of EEFs) in order to allow the planning threads to operate exclusively on the data. An advantage of running

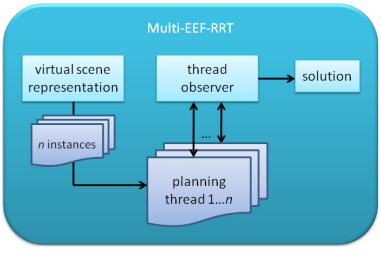

Fig. 4. The MultiEEF-RRT planner.

multiple planning threads in parallel is caused by modern CPU architectures that are multi-cored and thus each thread has almost the same performance than running alone on the system as long as the number of threads does not exceed the number of cores [16], [17], [18]. In case of a humanoid robot with two arms this advantage can already be exploited on dual-core machines.

Since the planning threads are executing probabilistically complete planners and by running multiple instances of these planners the probabilistic completeness is not affected, the *MultiEEF-RRT* planner is probabilistically complete.

D. Performance Analysis

The performance of the proposed MultiEEF-planner is analyzed by running the algorithms 30 times on a typical planning scene as depicted at Fig. 3b, consisting of a robot model, the kitchen environment, three obstacles and one target object with 10 associated grasping positions for each hand. The obstacles are located near the target object, resulting in limited free space to operate. The average planning time was measured with less than half a second and the time needed for smoothing the solution path was 0.7 seconds on average. The long runtime for path smoothing is caused by the high quality demands which can be reduced if a faster calculation is needed. In our performance evaluation 300 path smoothing cycles were performed, meaning that 300 C-Space shortcuts of the solution trajectory were searched. Since the MultiEEF-RRT planner generates collision-free IK solutions during the planning process, sometimes more than one IK solution has been calculated before a solution was found. As shown in table I, 1.3 IK solutions are computed during a planning run on average.

TABLE I
AVERAGE PERFORMANCE OF THE MULTIEEF-RRT PLANNER.

# IK Solutions	*Planning Time*	*Path Smoothing Time*	*Total Time*
1.3	449 ms	659 ms	1108 ms

V. VISUALLY CONTROLLED EXECUTION OF PLANNED TRAJECTORIES

The planned trajectories are executed by a component which keeps track of the current position and the speed of the joints. This module should be robust against error-prone sensor data as well as the inaccurate execution of velocities in order to offer a more robust execution. The quality of trajectory execution can be improved by visual observation and correction of the Cartesian poses. Using visual information to control the motion of the end effector is also referred to as visual servoing [19], [20] which is used for moving the EEF to a target position in general. In case of following a trajectory, the demands are slightly different, since the visual information is used for moving along a trajectory toward a specified target, which means that standard visual servoing techniques cannot be applied since these algorithms don't care about *how* the goal is reached, but try to reach the goal somehow. Our work on visually guided trajectory execution uses algorithms similar to PBVS approaches for controlling the trajectory execution with respect to tracked hand and object poses.

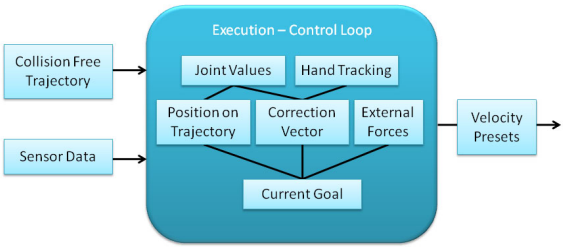

Fig. 5. The Control Loop of the trajectory execution.

Trajectories produced by RRT-based planning algorithms commonly consist of n dimensional support points connected by straight lines building a path through configuration space, where n is the number of degrees of freedom used for planning. Motion planning algorithms guarantee that all robot configurations located on the path lie in C_{free}, the collision-free part of the configuration space (sampling-based planning algorithms give this guarantee for a specified sampling resolution). Furthermore user-defined constraints, like end-effector orientations or maximal torques, can reduce the free C-Space. This makes planned paths rather fragile meaning that generally there is an unknown tolerance of leaving the path. To ensure the constraints are still satisfied when leaving the previously calculated path, the new path must be checked, which is mostly a too costly operation. Thus the module for executing planned C-Space trajectories has to take care of being as close to the path as possible.

In section V-A to V-D the general algorithm for following C-Space trajectories is described. This approach projects the current robot configuration to the nearest path segment and, depending on maximal allowed velocities in C-Space and workspace, the velocity values for the low-level controllers are calculated. We show that this module can handle noisy sensor data as well as changing loop times and inaccurate execution of demanded velocities. To improve robustness an approach for visually supporting the execution of planned C-Space trajectories is presented in section V-E. Here visual servoing approaches are used for controlling the position in workspace with respect to a planned path.

A. Follow a C-Space Trajectory

In this section the general control loop is explained without considering visual or force sensor data in order to explain the functionality of the algorithm. Further improvements of the approach are presented in the sections V-E and V-F where the information coming from the vision system and the force/torque sensors is used.

The algorithm is initialized with a C-Space trajectory and temporary goals are produced during execution. For that reason c_{robot}, the current configuration of the robot, is projected on the C-Space path, that consists of n dimensional straight line segments. Then, this projection c'_{robot} is used to determine if the actual temporary goal could be reached within the next looptime. If so, and the temporary goal was not the end of the path, the next C-Space goal is computed.

B. Projecting the Current Pose onto a Trajectory

1) Projection on a line segment: In order to determine current progress c_{robot} is projected on the collision-free C-Space path. Fig. 6a illustrates a two dimensional example where a configuration c is projected on a path segment p_1p_2 and the nearest configuration c' is depicted.

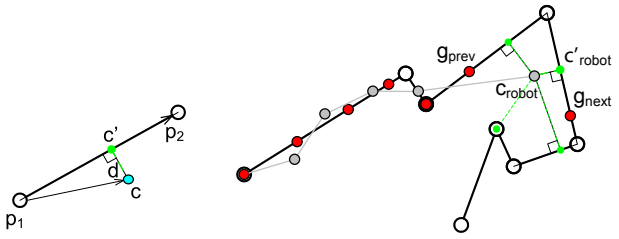

Fig. 6. A two dimensional example of projecting a configuration c onto a path segment (a) and on a complete path (b).

In general, the nearest configuration c' of a n dimensional configuration c to a path segment p_1p_2 can be calculated by computing the position r on the path segment (see equation 1 and 2).

$$r = \frac{p_1c \cdot p_1p_2}{p_1p_2 \cdot p_1p_2} \tag{1}$$

$$c' = \begin{cases} p_1, & r \leq 0 \\ p_2, & r \geq 1 \\ p_1 + r * p_1p_2, & 0 < r < 1 \end{cases} \tag{2}$$

All computations for determining the nearest path segment are based on the Euclidean metric. The algorithms could be easily extended by considering a more sophisticated metric which takes into account that an error of the shoulder joint has more impact than a displacement of a wrist joint.

2) Projection onto a trajectory: Fig. 6b illustrates how a configuration c_{robot} is projected onto a path that consists of multiple segments. In this figure the trajectory is marked by the white dots and the black line, the red dots show the temporary goals, which have been calculated in former steps, and the robot configurations according to sensor data are shown in gray. In the last loop the robot was supposed to move toward the temporary goal g_{prev} but the joint sensors say, the current configuration is at c_{robot}. Beginning with the segment of g_{prev} the segment with the shortest distance to c_{robot} is searched and the projection c'_{robot} is calculated like described before. In case of crossing paths this procedure guarantees that no path segment which lies behind g_{prev} is wrongly selected. For avoiding a misleading selection of future path segments a time horizon is used that limits the search of c'_{robot} to the next k segments.

C. Computing Temporary Goals

Temporary goals are used to determine the current direction of execution in C-Space and combined with the target velocity, the joint velocity values can be calculated. These goals are calculated as close as possible to c'_{robot}, the projected position on the path, to avoid cutting corners of C-Space path segments. They also have to be located further on the trajectory than the last temporary goal and also further than c'_{robot} to ensure that robot does not stumble.

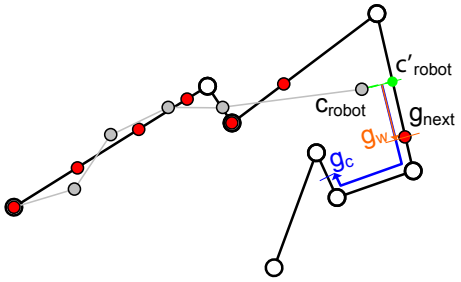

Fig. 7. Computing temporary goals.

A temporary goal is determined by starting at c'_{robot} and going along the path, predicting a future position on the trajectory. In case the previous goal g_{prev} lies further on trajectory than c'_{robot}, the prediction of future position starts at g_{prev} to prevent robot from stumbling. Depending on the maximal allowed velocity and the looptime two goal configurations g_c and g_w are calculated. These two goals describe the maximal reachable configurations on the path when considering the two parameters v_c, the maximal allowed motor velocity and v_w, the maximal workspace velocity for a joint. It is possible to define different v_c and v_w values for each joint, but in general the same value is used for all joints. When computing the goals, the maximal allowed position in the future is computed, so that the speed of no joint exceeds the maximal allowed velocities in C-Space and in workspace. In Fig. 7 the search for a new goal starts at c'_{robot} and both goal limits g_c and g_w are calculated. Although v_c would allow robot to move until g_c, this would violate the workspace velocity limit v_w which means that there would be a joint moving too fast. So g_w will become the next temporary goal g_{next}.

D. Generating Velocities

The direction of further movement in C-Space is given by $v = g_{next} - c_{robot}$, and in case v does not violate C-Space or workspace speed limits, it can be passed to the low level controllers.

In Fig. 8 the results of two simulation experiments are shown. The left figure shows the workspace TCP positions during the execution of the C-Space path. The white dots and the black line depict the path of the TCP in workspace and the red dots are placed at the workspace positions of the temporary goals which were computed during execution. The right figure shows the result of an experiment where an artificial lag of 1000 ms forces the high-level control loop to stop its execution. Since the low-level controller is still executing the last calculated velocity values, the TCP moves away from the trajectory. After the lag, the proposed algorithms are working as expected and the TCP is quickly moving back toward the path.

E. Visual Supervised Execution of C-Space Trajectories

The discussed components for executing C-Space trajectories are able to deal with changing loop frequencies and inaccurate execution of joint velocities. But in case the robot is not exactly registered in the world or camera and hand-eye calibration is not available or inaccurate, the execution

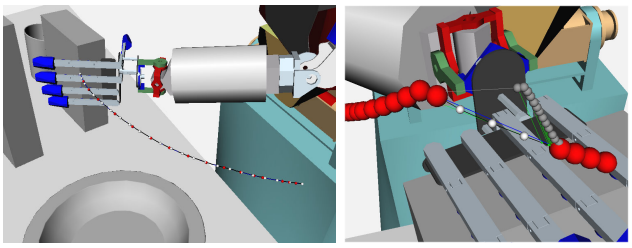

Fig. 8. (a) The workspace projection of a simulated execution of a C-Space path. The TCP follows the path (white dots) by moving the hand toward the temporary goals (red dots) which are computed during execution. (b) A narrow view of an executed trajectory where workspace equivalents of C-Space path points (white), goals (red) and the executed trajectory (gray) are depicted. The displacement of the TCP, caused by an artificial lag, is quickly compensated.

of C-Space trajectories will suffer from low accuracy. This is caused by the fact that even when the path is exactly executed, the corresponding positions in workspace will not match the ones in the virtual planning world. An exact planned trajectory will lead to a collision-free path in the virtual representation of the world, but in reality the difference between virtual world and reality can result in collisions. Thus a visual correction of the executed paths can help to retrieve more accurate TCP positions with respect to the surrounding. By visually controlling the execution of e.g. grasping trajectories a biological motivated principle is copied, since humans also use their eyes to support dexterous positioning of their hands.

To exploit as much information from the visual sensor channel as possible, the TCP and the target object are tracked continuously and their relative position to each other is utilized to correct the trajectory execution. Through this correction the TCP workspace pose is adjusted relatively to the target object, so that the visually determined relation matches the current relation in the virtual representation during path execution.

We use a hybrid estimation of the hand pose p^{vision}_{hand} by combining the tracked Cartesian hand position with the kinematically determined hand orientation. The hand tracking is realized by stereo localization of a red marker that is mounted on the hand [21]. Since the target and the hand poses are determined with the same sensor channel, the Cartesian relation

$$t^{vision}_{TcpToTarget} = p^{vision}_{target} - p^{vision}_{hand} \tag{3}$$

of these poses can be retrieved with high accuracy.

This relation can be calculated as well in the virtual world, by a determining

$$t^{virtual}_{TcpToTarget} = p^{virtual}_{target} - p^{virtual}_{hand} \tag{4}$$

In an exact calibrated system, the two relations $t^{vision}_{TcpToTarget}$ and $t^{virtual}_{TcpToTarget}$ should be identical. The displacement vector e is expressed by the difference between the virtual and the visually determined relations between target and TCP pose.

$$\boldsymbol{e} = \boldsymbol{t}^{virtual}_{TcpToTarget} - \boldsymbol{t}^{vision}_{TcpToTarget} \tag{5}$$

Since the pose in workspace is much more important than the C-Space configuration, we use e to build a C-Space correction vector Δq via the Jacobian's Pseudoinverse $J^{+}(q)$ (see algorithm 1). In case of violating joint limits or the Pseudinverse Jacobain calculations are trapped in local minima, the algorithm will fail to reach the desired position and the last valid value of Δq is used in further calculations.

$$\Delta \boldsymbol{q} = ComputeCSpaceDelta(\boldsymbol{e}, q) \quad (6)$$

Algorithm 1: $ComputeCSpaceDelta(\Delta x, q)$

```
x_start ← ForwardKinematics(q);
q' ← q; Δx' ← Δx;
while (!TimeOut()) do
    Δq ← kJ⁺(q')Δx';
    q' ← q' + Δq;
    x_new ← ForwardKinematics(q');
    Δx' = x_new − x_start;
    if (||Δx' − Δx|| ≤ threshold) then
        return (q' − q);
end
```

By moving the C-Space path by Δq during trajectory execution, the real TCP pose is adjusted so that the error in the relation between target and hand is reduced. The resulting trajectory will not be valid in the virtual planning surrounding any more (see Fig. 9), but in reality the displacements of the TCP can be reduced and a more precise execution is possible. Since Δq is calculated in each loop, the influence of static and dynamic errors is decreased.

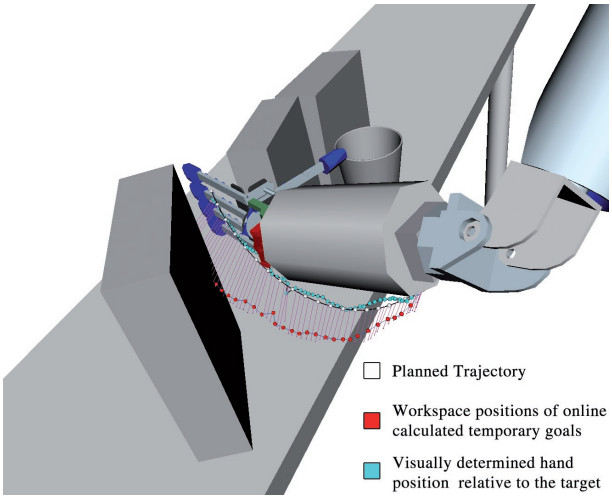

Fig. 9. A visually supported execution. The red lines represent the displacement of the original position on the path and the visually determined poses in workspace.

F. Including Reactive Components

Additionally to the proposed algorithms, the robot must be able to register unplanned contacts with the environment or humans and the system must react accordingly. Also interactions by a human operator, i.e. a correction of an executed trajectory, must be recognized and processed. Thus the component for execution of motions must comprise a reactive module that keeps track of contact or collision events with the environment.
Observing the sensor data coming from 6D force/torque sensors allows compliant reactions to contacts with the environment. If a force or torque is registered, the execution module reacts to this input and changes the movement of the joints to counteract these external forces. To guarantee a fast and secure behavior, this corrective behavior is proportional to the strength of the measured forces.

VI. Example with ARMAR-III

In this section we describe a realistic setup used for planning motions for grasping. The humanoid robot ARMAR-III is positioned in front of a sideboard and the goal is to grasp a known object for which a set of feasible grasps is stored in a database. The task includes the recognition and localization of the target object and obstacles to build up the internal representation of the scene (see sec. III), which then is used for planning a collision-free motion to reach the object with either the left or the right hand. Searching two paths in parallel is done by the *MultiEEF-RRT* planner as described in section IV. If a solution is found by the planner, the proposed *reactive execution framework* is parametrized and the planned trajectory is reliably executed. In Fig. 9 a planned trajectory is shown in workspace with white dots and the black line. During execution the visually determined hand position is used to calculate the displacement e, which is visualized by red lines. As shown the error between virtual world and reality changes during the execution of the trajectory. The workspace positions of the temporary goals are depicted as red dots and due to the Jacobian-based projection of the error from workspace to C-Space the displacement is not exactly represented, but a good approximation is served by the algorithms. The recognized TCP position with respect to the target object is marked by the green path which is close to the ideal planned trajectory.

The large deviations between the virtually planned and the executed trajectory, depicted as red lines in Fig. 9, are mainly caused by an inaccurate calibration of the joint sensor data. The joint position is derived from incremental sensors at the driving end, resulting in approximated joint values used as input for the control loop. The accumulated error of seven arm joints, one hip joint and three joints of the robot's neck causes large workspace displacements. Furthermore the active head is moving due to the actuated hip and neck joints. This means that a moving camera system produces a varying error in the transformation between the visually determined object poses and the global coordinate system.

Nevertheless, the humanoid robot ARMAR-III is able to execute the planned grasping trajectory with sufficient accuracy. Some pictures of the execution on ARMAR-III are shown in Fig. 10.

VII. Conclusion

In this work, we showed how a parallelized planning approach can be used to create grasping trajectories for multi

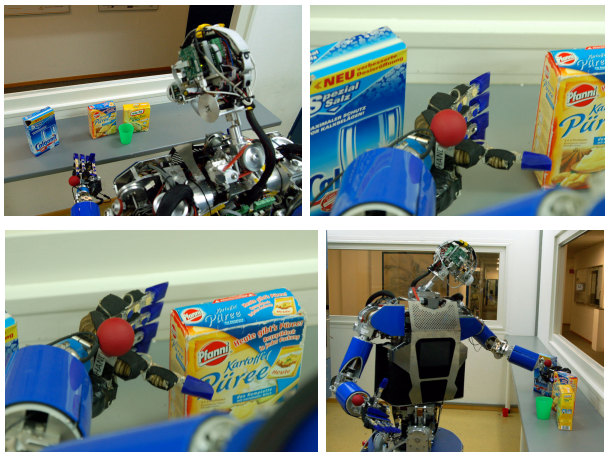

Fig. 10. ARMAR-III is executing a planned trajectory by the proposed visual servoing techniques.

arm robot systems like a humanoid robot. The *MultiEEF-RRT* planner calculates possible grasping configurations during the planning process and thus the planner is not limited to an incomplete set of IK solutions. Furthermore multiple grasps per object and end effector are used and thus the selection which grasp is reachable by a collision-free trajectory does not have to be done in advance since the planner implicitly selects a reachable grasp. It was showed by the experiments, that if one side of an object is blocked and grasping with one hand is impossible, the other hand is implicitly selected because the planner automatically generates a collision-free solution for one of the predefined grasps. Hence there is no need of a scene analysis and reasoning which hand should be used for grasping.

The presented execution component of ARMAR-III is able to deal with noisy sensor data, inaccurate executions of joint velocities, changing loop times and without any hand-eye calibrations. Through the visually observed execution the Cartesian position of the TCP is adjusted with respect to the (also visually localized) obstacles by continuously calculating a correction vector. By using the same sensor channel (the vision system) for building the virtual planning scene and for adjusting the execution, the errors between modeling and reality are reduced and the robot is enabled to execute planned grasping and manipulation trajectories.

Future work should address an improved hand tracking system to avoid the artificial marker and to allow the visual determination of hand orientations. Furthermore execution errors, like singularities in the Jacobian-based TCP movements, have to be managed by the system in order to offer a fail-safe execution even in unexpected situations.

VIII. ACKNOWLEDGMENTS

The work described in this paper was partially conducted within the German Humanoid Research project SFB588 funded by the German Research Foundation (DFG: Deutsche Forschungsgemeinschaft) and the EU Cognitive Systems project GRASP (IST-FP7-IP-215821) funded by the European Commission.

REFERENCES

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

[2] H. Sugiura, M. Gienger, H. Janssen, and C. Goerick, "Real-time collision avoidance with whole body motion control for humanoid robots," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, 29 2007-Nov. 2 2007, pp. 2053–2058.

[3] D. Berenson, R. Diankov, K. Nishiwaki, S. Kagami, and J. Kuffner, "Grasp planning in complex scenes," in *IEEE-RAS International Conference on Humanoid Robots (Humanoids07)*, December 2007.

[4] C. Eitner, Y. Mori, K. Okada, and M. Inaba, "Task and vision based online manipulator trajectory generation for a humanoid robot," in *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*, Dec. 2008, pp. 293–298.

[5] A. Nakhaei and F. Lamiraux, "Motion planning for humanoid robots in environments modeled by vision," in *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*, Dec. 2008, pp. 197–204.

[6] M. Gienger, M. Toussaint, and C. Goerick, "Task maps in humanoid robot manipulation," in *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*, Dec. 2008, pp. 2758–2764.

[7] F. Zacharias, C. Borst, and G. Hirzinger, "Capturing robot workspace structure: representing robot capabilities," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, 29 2007-Nov. 2 2007, pp. 3229–3236.

[8] M. Paulin, "Feature planning for robust execution of general robot tasks using visual servoing," in *CRV '05: Proceedings of the 2nd Canadian conference on Computer and Robot Vision*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 200–209.

[9] R. Swain Oropeza, M. Devy, and V. Cadenat, "Controlling the execution of a visual servoing task," *J. Intell. Robotics Syst.*, vol. 25, no. 4, pp. 357–369, 1999.

[10] P. Azad, T. Asfour, and R. Dillmann, "Stereo-based 6D Object Localization for Grasping with Humanoid Robot Systems," in *International Conference on Intelligent Robots and Systems (IROS)*, San Diego, USA, 2007.

[11] J. H. Reif, "Complexity of the mover's problem and generalizations," in *SFCS '79: Proceedings of the 20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*. Washington, DC, USA: IEEE Computer Society, 1979, pp. 421–427.

[12] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[13] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning." *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.

[14] S. LaValle and J. Kuffner, "Rapidly-exploring random trees: Progress and prospects." 2000, in Workshop on the Algorithmic Foundations of Robotics.

[15] N. Vahrenkamp, D. Berenson, T. Asfour, J. Kuffner, and R. Dillmann, "Humanoid motion planning for dual-arm manipulation and re-grasping tasks," in *Intelligent Robots and Systems, IROS*, October 2009.

[16] E. Plaku, K. Bekris, B. Chen, A. Ladd, and L. Kavraki, "Sampling-based roadmap of trees for parallel motion planning," *Robotics, IEEE Transactions on*, vol. 21, no. 4, pp. 597–608, Aug. 2005.

[17] S. Carpin and E. Pagello, "On parallel rrts for multi-robot systems," in *Proc. 8th Conf. Italian Association for Artificial Intelligence*, 2002, pp. 834–841.

[18] J. Cortes and T. Simeon, "Probabilistic motion planning for parallel mechanisms," vol. 3, Sept. 2003, pp. 4354–4359 vol.3.

[19] S. Hutchinson, G. Hager, and P. Corke, "A tutorial on visual servo control," *IEEE Trans. on Robotics and Automation*, vol. 12, no. 5, pp. 651–670, Oct. 1996.

[20] F. Chaumette and S. Hutchinson, "Visual servo control, part I: Basic approaches," *IEEE Robotics and Automation Magazine*, vol. 13, no. 4, pp. 82–90, dec 2006.

[21] N. Vahrenkamp, S. Wieland, P. Azad, D. Gonzalez, T. Asfour, and R. Dillmann, "Visual servoing for humanoid grasping and manipulation tasks," in *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*, Dec. 2008, pp. 406–412.