Grab a Mug – Object Detection and Grasp Motion Planning with the Nao Robot

Judith Müller Udo Frese Thomas Röfer
Deutsches Forschungszentrum für Künstliche Intelligenz, Cyber Physical Systems,
Enrique-Schmidt-Str. 5, 28359 Bremen, Germany
Email: {judith.mueller, udo.frese, thomas.roefer}@dfki.de

Abstract—**In this paper we introduce an online grasping system for the Nao robot [1] manufactured by Aldebaran Robotics. The proposed system consists of an object detector and a grasp motion planner. Thereby, known objects are detected by a stereo contour-based object detector and hand motion paths are planned by an A*-based algorithm while avoiding obstacles.**

Compared to skilled robots such as Justin [2] or ASIMO [3] online grasping with the Nao constitute as particular problem due to the limited processing power and the hand design. The methods proposed allow detecting and grasping objects in real-time on an affordable humanoid robot.

I. Introduction

Humans interact with their environment. Besides countenance, speech, and gestures, humans can also manipulate their surroundings with their hands. For instance humans can lift a coffee cup in order to drink. Since hands are one of the most useful human features, it seems that robots also could benefit.

In this paper we present an online grasping system that combines a stereo vision-based object detector and a grasp function for the affordable Nao robot [1]. The objects to grasp are ones that can be completely embraced by the robots finger and palm. We focus on small objects or objects with a handle, for instance a light standard-sized coffee cup or a pencil.

We use a Nao V3.3 with a stereo vision head that was specifically designed for us by Aldebaran Robotics (based on the Nao V4 head). In the design of the stereo vision, the two cameras are mounted parallel in the eye holes of the robot's head. The cameras are connected through an FPGA with an embedded PC mainboard equipped with an Intel Atom (1.6 GHz) processor. The acquisition of two images can be done in less than 33 ms. The recorded images overlap at approx. 10 cm outgoing of the center between the cameras.

There are many different ways to grasp an object, but not every option is the best. In the approach of Borst *et al.* [4], grasps are distinguished in force-closure and form-closure grasps. Force-closure grasps are able to balance disturbances by the wrenches (fingers) at the contact points. With form-closure grasps an object is completely embraced by the fingers and palm with the result that there is no friction on the contact points, even if the grasp holds.

Nao's hands are underactuated [5], because they are constructed with three flexible fingers per hand, which are controlled by a single motor (open / close). Unfortunately, the fingers are only really stiff if the hand is completely closed. Hence, experiments showed that solid objects such as coffee cups are only graspable if the grasp is form-closure. Furthermore, it seems that performing a force-closure grasp is not possible with this robot, since it is not able to move its fingers individually [6].

Most motion planners are operating either in Cartesian or in configuration space. While motion planners in configuration space as in [7] and [8] are able to guarantee a solution given there is one, planners in Cartesian space as in [9] are incomplete and difficult due to redundant kinematics. However, the integration of obstacle avoidance into a path planner operating in Cartesian space is simpler than in configuration space.

Furthermore, motion path planning in Cartesian space can be a very expensive process particularly when the grasping hand is attached to a humanoid robot, which can move in order to reach certain objects. Thus, the reachability needs to be checked by inverse kinematics for many points in order to select a suitable grasp and to validate reachability along the path. This process can be speed up best by a pre-calculated table as the capability map in [10] and [11]. By defining the workspace and storing information about how a region can be reached by the hand, it is possible to quickly select possible grasp points without the direct use of inverse kinematics. In our work we use the predefined workspace to solve redundant kinematics as well as the reachability along the motion path, which enables our motion path planner to quickly operate in Cartesian space. In contrast to the work of Cotugno and Mellmann [12], the grasping function proposed plans and executes only single handed grasps using an A*-based algorithm.

The first step of the general procedure of the grasp function proposed is to detect the object. Thereby edge detection is performed on the left and right stereo image by computing a contrast-normalized Sobel (cns) image instead of using color segmentation as in [13]. Afterwards the contrast images are

This work has been funded by the European Commission in the 7th Framework Programme for RTD in the context of the project 'ECHORD – European Clearing House for Open Robotics Development" under the contract number FP7-231143 and by DFG under grant FR2620/1-1 and SFB/TR 8 Spatial Cognition.

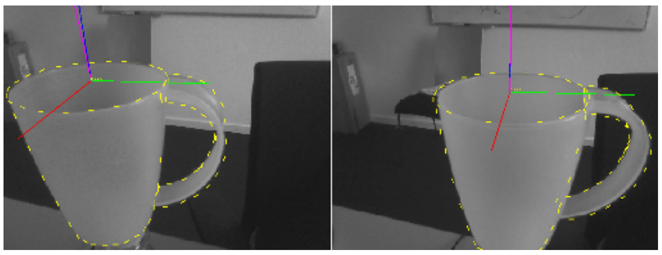

Fig. 1. A cup detected by our stereo contour-based object detector. The detection is based on a 3D model. Technically, the model contains a disc as cap which is specially labelled so its circle is always included in the contour.

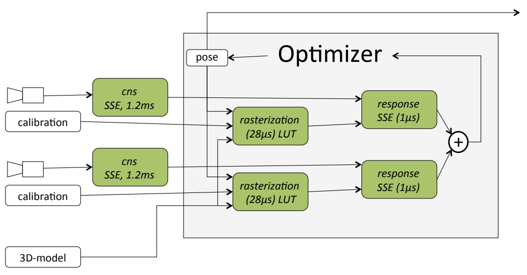

Fig. 2. Dataflow overview of our stereo contour-based object detector.

used to recognize the object by evaluating a range of possible object poses. The detected object is given to the grasp planner as a 6D pose.

The next step of the procedure is to decide whether an object can be grasped. By doing so, possible grasp hand poses are evaluated by using the pre-calculated workspace. Once a possible hand grasp pose is found, a motion path from the current hand position is planned. At last, the resulting motion path is executed by a trajectory based motion engine.

II. Stereo Contour based Object Response

On the perceptual side the goal was to recognize an object handed to the robot by a human in order to grasp it (Fig. 1), in our case a cup or a (big) pencil. There are many ways to do this and our motivation for the approach presented was the following: In computer vision there is the general insight that taking hard decisions early impairs robustness. Examples therefor are pixel-wise color segmentation or Canny edge detection followed by line segment extraction followed by object detection. Instead, one should take a decision only after considering all relevant input data, in our case the whole stereo image, assessing which interpretation is overall most supported by the data. Compared to mono, stereo gives a better depth perception, and following the above paradigm we do a combined search in both images, not separately. This is of course computationally expensive and here we wanted to investigate, whether this thorough methodology can be implemented for object detection on Nao's limited computation power.

A. *Overview*

Our detection is contour-based, so following the above methodology, the detector considers a range of hypotheses for the object pose and evaluates for each how much it is supported by the stereo image, i.e. how much the image at those points where a contour should be looks like an actual contour. Consider Figure 2 ignoring the "cns"-box for the moment. The first step is rasterization, i.e. rendering the object in a given hypothetical pose from the perspective of the left and right cameras. The result is a 2D-contour, i.e. a function $[0 \dots 1] \to \mathbb{R}^2$. The second step is contour evaluation, i.e. computing a response how much the contour is supported by the image. Its definition has been adapted from our previous work [14], where it is used for circle detection. Based on this goal function, an optimizer searches through the space of possible poses, finding the cup pose with the largest response.

B. *Contour Response*

The contour response $\in [0 \dots 1]$ is naturally defined as an integral over the contour

$$\operatorname{cresp}(p) = \int_0^1 \operatorname{resp}\left(p(\lambda), \angle p'(\lambda) + \frac{\pi}{2}\right) d\lambda. \tag{1}$$

The response $\operatorname{resp}\left(p(\lambda), \angle p'(\lambda) + \frac{\pi}{2}\right)$ at a single contour point $p(\lambda)$ indicates, how much the local image at $p(\lambda)$ looks like the ideal contour, i.e. an intensity gradient in the direction $\angle p'(\lambda) + \frac{\pi}{2}$ normal to the contour. Additionally, we demand linear illumination invariance from resp and specify what "local" means by the 3×3 weighting mask $\frac{1}{16}\begin{pmatrix} 1&2&1\\2&4&2\\1&2&1 \end{pmatrix}$. From these assumptions, it follows that

$$\operatorname{resp}(x, y, \theta) = \frac{2\left(X * \left[\cos\theta \begin{pmatrix} 1&0&-1\\2&0&-2\\1&0&-1 \end{pmatrix} + \sin\theta \begin{pmatrix} 1&2&1\\0&0&0\\-1&-2&-1 \end{pmatrix}\right]\right)^2}{X^2 * 16\begin{pmatrix} 1&2&1\\2&4&2\\1&2&1 \end{pmatrix} - \left(X * \begin{pmatrix} 1&2&1\\2&4&2\\1&2&1 \end{pmatrix}\right)^2} \begin{pmatrix} x\\y \end{pmatrix}, \tag{2}$$

where X is the image and $*$ denotes convolution. The Sobel filters in the nominator follow from the choice of the weighting mask. The denominator is a weighted image variance that follows from illumination invariance. The details of this derivation are beyond scope here, but the fact that (2) follows from first principles support our methodical claim.

C. *Contrast Normalized Sobel (CNS) Image*

Equation (2) needs to be evaluated for many points along many contours. Fortunately, it can be decomposed

$$\operatorname{cns} = \frac{\sqrt{2}\left[X * \begin{pmatrix} 1&0&-1\\2&0&-2\\1&0&-1 \end{pmatrix}, X * \begin{pmatrix} 1&2&1\\0&0&0\\-1&-2&-1 \end{pmatrix}\right]^T}{\sqrt{X^2 * 16\begin{pmatrix} 1&2&1\\2&4&2\\1&2&1 \end{pmatrix} - \left(X * \begin{pmatrix} 1&2&1\\2&4&2\\1&2&1 \end{pmatrix}\right)^2}} \tag{3}$$

$$\operatorname{resp}(x, y, \theta) = \left(\begin{pmatrix} \cos\theta\\ \sin\theta \end{pmatrix}^T \operatorname{cns}(x, y,)\right)^2 \tag{4}$$

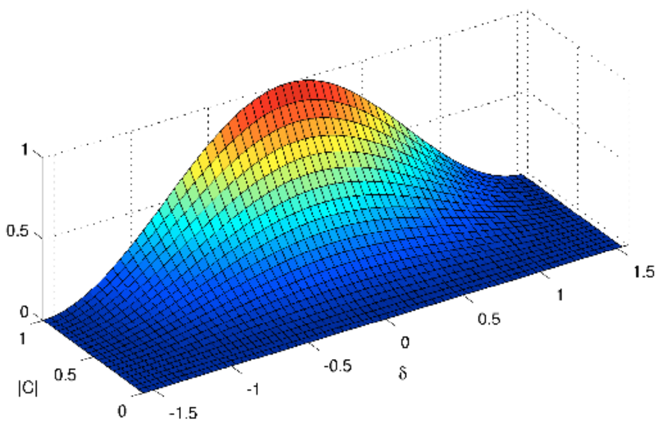

Fig. 3. Plot of the response (5) as a function of cns norm $|\operatorname{cns}(x,y)|$ and angular mismatch $\delta = \theta - \angle\operatorname{cns}(x,y)$. The corresponding plot for the response used in the classical Hough-transform is 1 inside a hand-tuned rectangle and 0 outside.

into the squared scalar product of the contour-normal $\left(\begin{smallmatrix}\cos\theta\\ \sin\theta\end{smallmatrix}\right)$ and a contrast-normalized Sobel image (cns). The cns-image does not depend on the contour and is only computed once (Fig. 2). The contour-normal is reused when searching at different positions and only the squared scalar product needs to be computed for every point of every contour evaluated.

Also, (4) has a nice interpretation, as

$$\operatorname{resp}(x,y,\theta) = \cos^2\left(\theta - \angle\operatorname{cns}(x,y)\right)|\operatorname{cns}(x,y)|^2. \qquad (5)$$

This means that the detector gradually penalizes an angular mismatch between the desired and the observed gradient direction as well as local images which do not look gradient-like as indicated by the cns-norm (Fig. 3). Since the latter is illumination invariant, it measures “gradient purity” as opposed to “gradient intensity” for the classical Sobel vector.

D. Efficient Implementation

The cns-computation (3) has been implemented using Single Instruction Multiple Data (SIMD) parallelism, here Intel's SSE3 instruction set. Multi-core parallelism would also be easy, but is not available on Nao's Atom processor. Also the various filters were separated into $X * \left(\begin{smallmatrix}1 & 2 & 1\end{smallmatrix}\right) * \left(\begin{smallmatrix}-1 & 0 & 1\end{smallmatrix}\right)^T$, $X * \left(\begin{smallmatrix}-1 & 0 & 1\end{smallmatrix}\right) * \left(\begin{smallmatrix}1 & 2 & 1\end{smallmatrix}\right)^T$ and $X * \left(\begin{smallmatrix}1 & 2 & 1\end{smallmatrix}\right) * \left(\begin{smallmatrix}1 & 2 & 1\end{smallmatrix}\right)^T$, reusing the horizontal $X * \left(\begin{smallmatrix}1 & 2 & 1\end{smallmatrix}\right)$ intermediate result. In order to maximize parallelism, the filters are computed in 16 bit, converting to float for the reciprocal square root only. Also, a regularizer of 256 (1 intensity unit) is added in the square root to avoid $0/0$.

The actual response evaluation (4) is most time-critical. Intel's SSE3 includes an instruction `pmaddubsw` that multiplies 2×16 bytes adding adjacent products (16 bit). This computes $\left(\begin{smallmatrix}\cos\theta\\ \sin\theta\end{smallmatrix}\right)^T \operatorname{cns}(x,y)$ in (4). However, one operand is unsigned, so we shift the cns image by 1, (technically 128), making it positive and correct the product by subtracting $\cos\alpha + \sin\alpha$. The result is squared (4) and accumulated (1). Overall, 6 instructions perform 8 evaluations of (2). The evaluations correspond to an identical contour but 8 successive pixels in the (cns-) image. To exploit this parallelism, we always compute 8×8 (or even 16×16) responses of the same contour shifted by $[0 \dots 7] \times [0 \dots 7]$.

III. 3D Object Search Process

A. Rasterization

The rasterization (Fig. 2) takes a triangle mesh as 3D object model, a camera calibration and a hypothetical object pose as input and renders the contour of the object at the given pose as viewed from the camera. The first step is to determine which edges of the model form the contour. At the moment, we go through all edges and select those where one adjacent face is viewed from the front and one from the back. This does not consider global occlusion, an extension that could be implemented in the future. As a special rule, faces can be marked by a color label and edges between visible faces of different labels are also added. This feature is needed to make for instance the front edge in Figure 1 part of the contour.

Contour-edge determination is expensive. Fortunately, it depends only on the *camera's* position *relative to the object*, not its orientation, so we precompute a 3D look-up-table.

Next, the vertices involved are perspectively projected into the image in an SSE implementation. The projection ignores distortion which is ≈ 1 pixel only for Nao. The precomputed edge list is sorted such that projected vertices can be used twice. In principle, one would raster each edge, e.g. with the Bresenham line-algorithm, then. However, to save computation, time we only use the mid-point of each edge in the contour and compute the contour orientation at that point from the two projected vertices. Instead of θ, we directly compute $\sin\theta$ and $\cos\theta$ to avoid expensive trigonometrical operations. Midpoints outside the image are skipped.

Overall, with these optimizations, computation time is reduced from $2 \times 600\mu s$ to $2 \times 28\mu s$.

B. Local Search (Refinement and Tracking)

During local search, the optimizer (Fig. 2) changes the pose towards growing responses. This procedure is used for tracking as well as to refine a coarse initial pose obtained by our global search heuristic presented later. We use the simple approach to optimize DOFs round-robin one at a time, although there are of course more sophisticated optimization algorithms. However, we exploit that the response computation provides an array of 8×8 responses for shifted contours (2D translation).

So, to refine one DOF, we compute 8×8 responses around the original pose, around a pose changed on step in the considered DOF, and around the inversely changed pose. The subpixel-refined maximum of these $3 \times 8 \times 8$ responses defines the new pose. Therefor the image translation must be converted into a change of pose. This is approximated by a rotation of the object around the camera which moves the object's center in the image according to the obtained image translation.

As image translation is already covered, the 4 remaining DOFs are translation in viewing direction and object rotation around X, Y and Z (skipped in case of symmetry). The step size is roughly determined to create 3 pixel changes in the image based on object size and distance.

At first sight it may appear to be a waste of time to refine image-translation each time one of the four DOF is refined. However, typical object movement creates large image movement but rather small change in shape, so this approach makes the system capable of tracking faster movements and exploits the SSE-based 8×8 block computation.

Finally, one remarkable finding was that the stereo contour-based object response has a rather large range of convergence as shown in Figures 7 and 9 in Section V. This supports our motivation that avoiding early hard decisions improves robustness.

C. Global Search Heuristic

The textbook solution for global object search would be to find the maximum response of all poses within the grasping space (6DOF). However, this is computationally beyond scope despite all the optimizations mentioned. Instead, we use an application-specific heuristic. We search only for a single cup orientation by assuming it is roughly vertically aligned and by removing the handle, making it rotationally symmetric. This orientation is obtained from the robot's forward kinematic and the decision to use only one orientation was motivated by the large range of convergence mentioned above.

For the position, we go through the image in patches of 64×48 pixels and rasterize the cup at several positions along the center pixel's ray. For each contour 64×48 responses are computed and the largest overall response is refined.

Then cup-rotation is determined by evaluating the response of several rotated cups with handle. Finally, the full model is refined and if the response exceeds a threshold (0.65) we switch to tracking mode. If the response in tracking mode falls below 0.5 for 15 frames, we switch back.

The global search takes ≈ 320ms, so we spread it over several frames, evaluating only between one and two 64×48 blocks in each frame ($13-26$ms).

IV. Object Grasping

On the motion side, the goal is to successfully grasp an object. More specific, the goal is to decide from which hand pose the object can be grasped and to find a valid motion path from the current hand pose. In addition, obstacles such as the object itself or body parts that may be in the direct path between the target position and the start position have to be avoided.

We decided to implement a motion planner instead of a reactive approach in order to foresee collisions with the object or parts of the robot body. Since Nao's processor is limited as well as the degrees of freedom (DOF) of Nao's arms, we decided to pre-calculate the workspace and to use this table in the motion planner for efficiency.

A. Overview

While most robotic arms have six or even seven DOF, Nao's arms are constructed with only five DOF. The fifth DOF represents the wrist angle and has only minor influence on the planning. So our approach ignores this DOF first, using only four DOF to define a certain hand pose, and handles the wrist-DOF later. This leaves only one DOF of four for the lower arm direction while a fixed hand position is commanded, less than the two possible DOF. According to that, the reachability of Nao's hand is clearly very limited and the lower arm direction depends on the hand position. For that reason it is necessary to check for each grasp pose and each point on a motion path whether it can be reached. This leads to the problem that a large number of reachability checks are necessary for motion planning. This can be speed up best by predefining the workspace in a reachability map. The reachability map is precomputed and used as the basis of the whole grasping process. The first step is to evaluate a range of grasp poses by using the map. Once a reachable grasp pose is found, the grasp planner plans a path through the grid cells of the map. The planning procedure is to evaluate nodes starting with the goal node (grasp pose) in order to find a path to the current hand pose. For each node to be evaluated possible collisions are inspected. The resulting motion plan is converted into a Bezier path and executed by a trajectory-based motion engine. During the motion execution, the motion plan is updated in each frame.

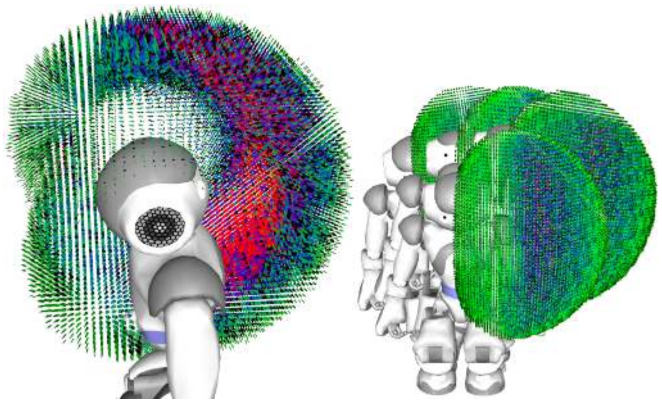

Fig. 4. Left: The best reachable regions are marked in red, less well-reachable regions are marked in blue and badly reachable regions are marked in green. Right: Extended reachability map due to multiple possible shoulder positions: sit down, lean left, lean right, and stand.

B. Reachability Map

In our approach we discretize the workspace with a cube that is divided into equally sized smaller cubes. Each sub cube serves as a region in the workspace. Each region stores a set of reachable lower arm directions (1 DOF). Because the wrist rotation can be calculated later from the lower arm direction and the joint limits, we only need to store a set of possible lower arm directions per region instead of a set of full hand orientations. The left side of Figure 4 pictures the

reachability map used, where only reachable directions per region are marked.
The map is created offline by using forward kinematics for each 0.5 degree angle of each arm joint. In that process we calculate the position of the palm as well as the direction of the lower arm and mark them in the map. In order to keep the memory footprint as small as possible, each lower arm direction calculated is rounded to a set of 512 directions, which are generated by using the spiral point algorithm proposed by Saff and Kuijlaars [15].
The origin of the reachability map is located in the shoulder of the robot. Thus, it is possible to test with different shoulder positions whether a certain hand position is reachable without the use of inverse kinematics. The right side of Figure 4 pictures how the workspace increases with only four possible shoulder positions.

C. Grasp Selection

Before the motion plan can be constructed, a target position needs to be selected. In order to do this, we assign a set of predefined grasp rules to each object. Each rule is defined by a grasping point and a range of lower arm directions relative to the object. In Figure 5, grasp rules are marked with blue triangles. The green dots constitute the position where to grasp and the triangle defines a range of lower arm directions.
In order to select a grasp, the grasp rules are matched with the reachability map defined in Section IV-B. In this process, areas that include a grasping point are examined further in order to check whether the corresponding possible lower arm directions are qualified for the grasp. This step is necessary, because the possible lower arm directions per area are very limited (Fig. 4). In this process, the possible lower arm directions (red in Fig. 5) of the grasp areas are compared to the angle ranges from the grasp rules (matching directions are shown in yellow in Figure 5). The best match is selected.

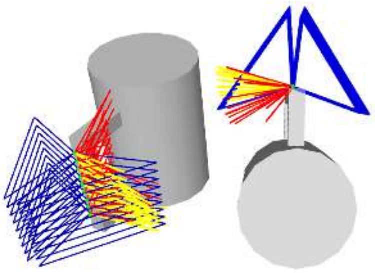
Fig. 5. Each object has its own grasp map, which is generated from a set of predefined grasp rules. Each rule connects a range of lower arm directions to a grasp position (blue). The reachability map (red) is matched to the object's grasp map. Matches are marked yellow.

D. Motion Planning

The next step is the actual planning from the selected grasp position (start node) to the current hand position (goal node). Compared to just using linear joint angle interpolation

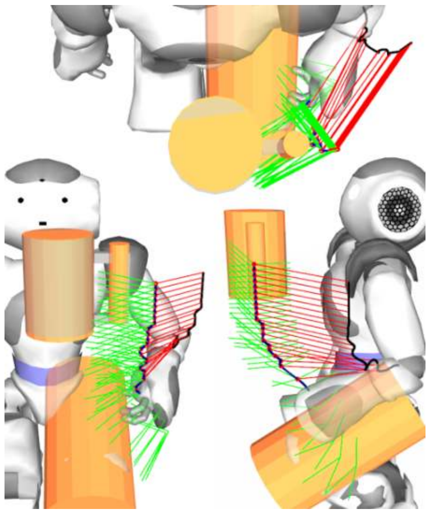
Fig. 6. The linear interpolated motion path of the hand and the elbow. Lower arm directions per waypoint are represented with red lines. Obstacles are marked by orange cylinders and the finger positions calculated per waypoint are represented by green lines.

this allows to avoid collisions on the way. The planner uses the precomputed reachability map without the need for reachability checks via inverse kinematics.
The reachability map provides the planner with 6D information on the possible hand positions and lower arm directions. Since planning in 6D is very expensive, our A*-based planning algorithm initially only uses the 3D area grid and ignores the lower arm direction and hand orientation. Thereby, to be evaluated, nodes are checked for reachability and obstacle collision in order to calculate the heuristics only for verified nodes. In this process, nodes with more suitable lower arm directions are rated better than nodes with greater deviations from the lower arm goal direction. Also the distance between the node evaluated and the goal node in 3D are taken into account.
The output from the planning algorithm is a list of waypoints through the reachability map, which are represented as red dots in Figure 6. Since there is a dependence between the hand positions the directions of the lower arm, a waypoint also includes a direction. Each direction defines the elbow position corresponding to the waypoint and is marked by red lines in Figure 6. In order to execute a plan found, a trajectory-based motion engine [16] was extended to take arm movements as input. Since the grasp plan consists of waypoints, executing them with linear interpolation would exhibit velocity discontinuities at the waypoints. In order to overcome this problem, the grasp plan is converted into a Bezier spline using the method by DeRose *et al.* [17].

1) A Heuristic and A* Cost:* The A* heuristic estimates the costs to reach the goal from the current node, while the A* cost function calculates the costs from the start node to the current node. If an admissible heuristic is used, A* finds the optimal way from the start node to the goal node.
In equation (6), the heuristic h_σ for a node x is calculated

as the weighted sum of the position difference (7) and the difference in lower arm direction (8) with f_{h_0} and f_{h_1} as factors (cf. Table I).

$$h_\sigma = f_{h_0} h_0 + f_{h_1} h_1 \tag{6}$$

$$h_0 = |x - x_{goal}|; \tag{7}$$

$$h_1 = |d(x) - d(x_{goal})|; \tag{8}$$

Thereby not only the distance to the goal node but also the change in the direction of the lower arm $(d(x_i))$ is considered. This is necessary, because most nodes are only reachable with a few possible lower arm directions as it is shown in Figure 4. If the heuristic would only take the distance into account, there could be nodes on the path, in which the lower arm directions are not matching to one another. This could lead to an inhomogeneous motion path. The aim of this approach is to add an extra weight to each node, which rewards more homogenous nodes. Hence, a closer area with a badly rated lower arm direction is valued inferior to an area with a well-rated lower arm direction.
Analogously to the heuristic, the cost-function (9) sums the distance of one node to the next as well as the changes in the direction of the lower arm.

$$c_\sigma = f_{c_0} c_0 + f_{c_1} c_1 \tag{9}$$

$$c_0 = \sum |x_k - x_{k-1}|; \tag{10}$$

$$c_1 = \sum |d(x_k) - d(x_{k-1})|; \tag{11}$$

2) Obstacles: The object to grasp as well as the body parts such as fingers and legs are included into the planner as obstacles. Since collision checks can be very expensive, we only test whether the fingers collide at this time. Furthermore, we only use cylinders (orange in Fig. 6). In doing so, all obstacle positions except for the fingers are calculated once per frame. In contrast, finger positions are calculated for each node that is to be evaluated. These finger positions are represented by green lines in Figure 6.
In order to check whether any part of a finger collides with the object or another limb, the shortest distance or the intersection, respectively, between each obstacle and each finger is calculated. Thereby, if the shortest distance between a finger and an obstacle is smaller than the sum of obstacle and finger radius, a collision is detected. In that case, the corresponding node is rejected as possible waypoint of the path.

V. Experiments

All experiments were made on a Nao robot using its Intel Atom (1,6 GHz) processor with 1 GB SDRAM.

A. Stereo Contour-based Object Response

We have evaluated the contour-based stereo detector on a set of 53 images taken in a cluttered office environment. Figure 7 shows a typical image with the rough pose detected by the global search and after refinement. Figure 8 shows a roc-curve of the detector. In our opinion the performance is good given the highly cluttered scenes and the fact that often the cup is only partially visible in the image and partially occluded by the hand. Figure 9 shows that the detector has a rather large range of convergence, which allowed us to perform the global search efficiently with a rather coarse grid and only a single orientation.

Computation time of the detector is $2 \times 1.2ms$ for the cns-computation, $28\mu s$ for rasterization of one pose in one camera and $1\mu s$ for response evaluation of one contour, when always blocks of 8×8 contours are evaluated.

B. Planning

We tested the planning algorithm with different heuristic and cost-function parameters for the equations (6) and (9) as shown in Table I. The frame rate of the planner was 30 Hz. The first parameter set from Table I generates an admissible heuristic and a cost function that prefers motion plans with

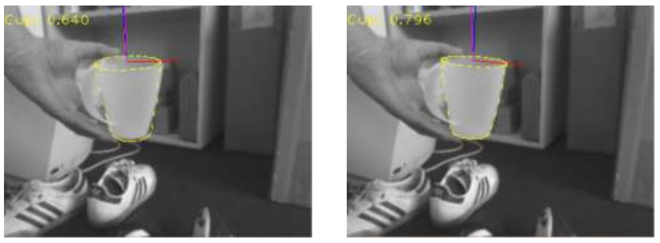

Fig. 7. An example of a global search result before and after refinement.

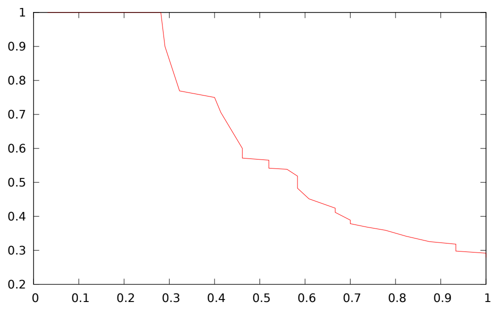
Fig. 8. Precision over recall (roc-curve) for the stereo image based cup detector.

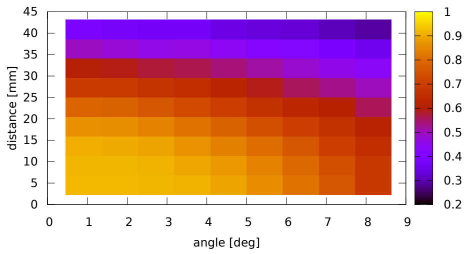
Fig. 9. Probability with which local refinement converges to the true pose as a function of the angular and translational distance between starting and true pose. The probability is computed with 100 tries in each image. The cup is $95 \times 75mm$ large, so nearly half a cup-diameter in the initial guess leads to a good final pose.

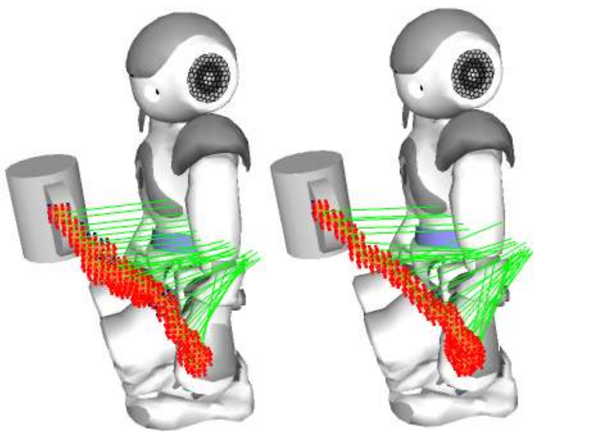
Fig. 10. Left: Depiction of nodes evaluated by using the first parameter set from Table I. Nodes considered are represented as dots. The path planned and the corresponding lower arm directions per waypoint are marked by lines. Right: Depiction of nodes evaluated by using second parameter set from Table I.

homogenous lower arm directions, which is shown in the left half of Figure 10. This heuristic takes the distance between the current and the goal node as well as the difference between the current and the goal node lower arm direction into account. The cost function summarizes the distance between the current node and its predecessor as well as the difference between the current and the predecessor's lower arm direction. Although many nodes need to be processed, this parameter set is usable for real-time grasping.
The second parameter set from Table I generates an admissible heuristic and cost function that takes only the distances into account. The heuristic considers the distance between the current and the goal node and the cost function summarizes the distance between the current node and its predecessor. In comparison, the parameter set compels that more nodes need to be processed and consequently, the computation is almost 50% slower. Since only nodes that are closer to the target node are rated superior, some elusive nodes are included into the path. This is especially the case around the lower leg, which is included to the planner as an obstacle. Furthermore, on the right side in Figure 10, the problem of inhomogeneous motions is conspicuous.
With the last two parameter sets from Table I we only consider the heuristic and ignore the cost function. This follows the approach of a greedy-search, whereby only the estimated costs are considered to select a successor node. We anticipated a lower calculation time but a less optimal path,

TABLE I
HEURISTIC AND COST-FUNCTION PARAMETERS IN COMPARISON TO CALCULATION TIME PER FRAME.

f_{c_0}	f_{c_1}	f_{h_0}	f_{h_1}	$average\frac{ms}{frame}$
1	0.5	1	0.5	20 ms
1	0	1	0	29 ms
0	0	1	1	13 ms
0	0	1	0	14 ms

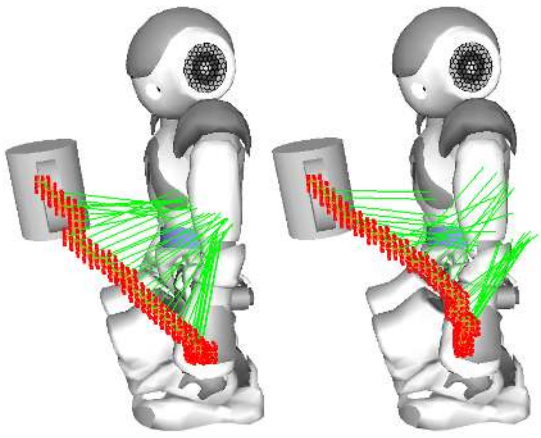
Fig. 11. Left: Depiction of nodes evaluated by using the third parameter set from Table I. Right: Depiction of nodes evaluated by using the last parameter set from Table I. Planning is done from the object to the hand.

which actually is the case.
According to that the left half of Figure 11 constitutes the third parameter set from Table I tested. The corresponding non-admissible heuristic considers the difference of the lower arm direction in comparison to the goal node as well as the distance. The result is not the shortest but a homogenous path. In comparison to the first parameter set, the calculation time could be lowered by about almost one third. The last parameter set from Table I also generates a non-admissible heuristic but only considers the distance to the goal node. As it can be seen on the right half of Figure 11 the path is inhomogeneous and no real speed difference can be measured compared to the third parameter set.
We also made experiments regarding the correct solution of the planning algorithm with obstacles. Therefore, we placed a cylindrical obstacle in between the robot and the object to grasp. Figure 12 depicts the correct solution of the planning algorithm. Since the workspace is rather small there are very few positions where an additional obstacle can be placed. Hence, additional objects often obstruct the workspace such that no solution can be found.

C. System Level Experiments

In 30 experiments, we tested whether a cup could be grasped and how long it took. Thereby, we recorded the time between the first detection and the successful grasp in a normal illuminated simple office environment. In 29 trials, the cup could be successfully grasped with an average duration of 10.026s. The duration is Gaussian distributed with a standard degression of 1.42s as it can be seen in Figure 13. As a consequence, the grasp time is with a probability of 95% between 7.186s and 12.866s.
During the trials we also measured the timings of the object detector. Both cns images are calculated with an average time of 2.5ms per frame, the global search with 21.6ms per frame and the refinement in 1.8ms per frame. An exemplary video of the overall system accompanies the paper.

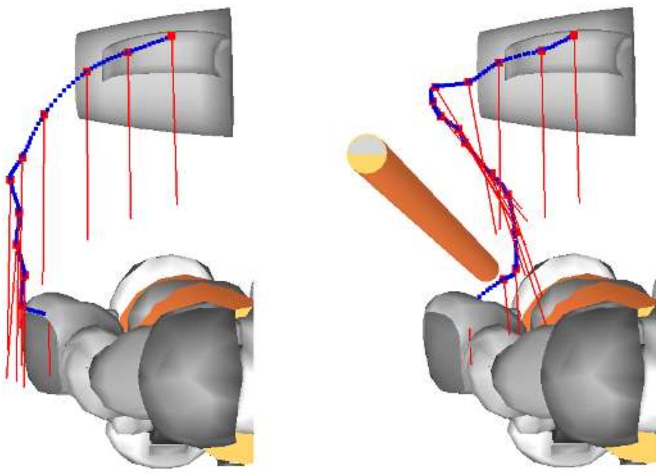

Fig. 12. Topview of the smoothed Bezier path with only the object to grasp as obstacle (left) and with an additional obstacle (right).

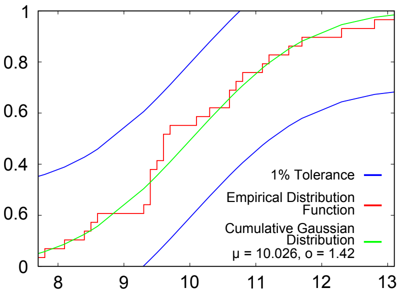

Fig. 13. We performed the Kolmogorov-Smirnov test with a confidence level of 0.1. The result proves with a probability of 99% the hypothesis of Gaussian distributed measurements.

VI. Conclusion

Summarizing, the online grasping system proposed is not only able to plan and execute a grasp on the affordable Nao robot, but it also is fast enough for a frame rate of 30 Hz. It seems that a precomputed reachability map can have very positive effects on the planning performance in such a way that a sufficient amount of possible waypoints can be processed. Furthermore, the search space can be decreased by including the evaluation of the lower arm direction into the heuristic as well as into the cost function. By using a non-admissible heuristic the planning speed can be increased, which enables robots (i.e. older Nao models) with less processor power to also use the proposed grasp function.
Furthermore, we were able to analyze Nao's new stereo vision design and we discovered that the overlap of the cameras is sufficient to detect objects very near to the head. However, since the motion range of Nao's head pitch joint is not high enough to view the area directly in front of its body the overlap of what can be detected and what can be grasped is rather small.

Acknowledgments

We would especially like to thank Oliver Birbach for calibrating the cameras of our Nao.

References

[1] D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. Lafourcade, B. Marnier, J. Serre, and B. Maisonnier, "Mechatronic design of nao humanoid," in *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, may 2009, pp. 769 –774.

[2] C. Borst, T. Wimbock, F. Schmidt, M. Fuchs, B. Brunner, F. Zacharias, P. R. Giordano, R. Konietschke, W. Sepp, S. Fuchs, C. Rink, A. Albu-Schaffer, and G. Hirzinger, "Rollin' justin - mobile platform with variable base," in *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, may 2009, pp. 1597 –1598.

[3] 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*, vol. 3, 2002, pp. 2478 – 2483 vol.3.

[4] C. Borst, M. Fischer, and G. Hirzinger, "Grasping the dice by dicing the grasp," in *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 4, oct. 2003, pp. 3692 – 3697 vol.3.

[5] K. Lalibertže, L. Birglen, and C. M. Gosselin, "Underactuation in robotic grasping hands," in *Journal of Machine Intelligence and Robotic Control*, vol. 4, no. 3, 2002, pp. 77 –87.

[6] G. Kragten, A. Kool, and J. Herder, "Ability to hold grasped objects by underactuated hands: Performance prediction and experiments," in *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, may 2009, pp. 2493–2498.

[7] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566 –580, aug 1996.

[8] K. Harada, K. Kaneko, and F. Kanehiro, "Fast grasp planning for hand/arm systems based on convex model," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, may 2008, pp. 1162 –1168.

[9] N. Vahrenkamp, T. Asfour, and R. Dillmann, "Simultaneous grasp and motion planning: Humanoid robot armar-iii," *Robotics Automation Magazine, IEEE*, vol. 19, no. 2, pp. 43 –57, june 2012.

[10] F. Zacharias, C. Borst, and G. Hirzinger, "Object-specific grasp maps for use in planning manipulation actions," in *Advances in Robotics Research*, T. Kröger and F. M. Wahl, Eds. Springer Berlin Heidelberg, 2009, pp. 203–213.

[11] ——, "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.

[12] G. Cotugno and H. Mellmann, "Dynamic motion control: Adaptive bimanual grasping for a humanoid robot," in *Proceedings of the Workshop on Concurrency, Specification, and Programming CS&P 2010*, vol. Volume 2, Börnicke (near Berlin), Germany, September 2010.

[13] P. Azad, T. Asfour, and R. Dillmann, "Stereo-based 6d object localization for grasping with humanoid robot systems," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, 29 2007-nov. 2 2007, pp. 919 –924.

[14] O. Birbach, U. Frese, and B. Bäuml, "Realtime perception for catching a flying ball with a mobile humanoid," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China*, 2011.

[15] E. Saff and A. Kuijlaars, "Distributing many points on a sphere," *The Mathematical Intelligencer*, vol. 19, pp. 5–11, 1997.

[16] J. Müller, T. Laue, and T. Röfer, "Kicking a ball - modeling complex dynamic motions for humanoid robots," in *RoboCup 2010: Robot Soccer World Cup XIV*, ser. Lecture Notes in Artificial Intelligence, J. R. del Solar, E. Chown, and P. G. Ploeger, Eds., vol. 6556. Springer; Heidelberg; http://www.springer.de/, 2011, pp. 109–120.

[17] T. D. DeRose and B. A. Barsky, "Geometric continuity, shape parameters, and geometric constructions for catmull-rom splines," *ACM Trans. Graph.*, vol. 7, no. 1, pp. 1–41, Jan. 1988.