Visual Servoing on Unknown Objects

Xavi Gratal, Javier Romero, Jeannette Bohg, Danica Kragic

Computer Vision and Active Perception Lab
Centre for Autonomous Systems
School of Computer Science and Communication
Royal Institute of Technology, 10044 Stockholm, Sweden
{javiergm,jrgn,bohg,danik}@nada.kth.se

Abstract

We study visual servoing in a framework of detection and grasping of unknown objects. Classically, visual servoing has been used for applications where the object to be servoed on is known to the robot prior to the task execution. In addition, most of the methods concentrate on aligning the robot hand with the object without grasping it. In our work, visual servoing techniques are used as building blocks in a system capable of detecting and grasping unknown objects in natural scenes. We show how different visual servoing techniques facilitate a complete grasping cycle.

Keywords: visual servoing, object grasping, calibration, active vision

1. Introduction

Object grasping and manipulation stands as an open problem in the area of robotics. Many approaches assume that the object to be manipulated is known before hand [1, 2, 3, 4]. If the object bears some resemblance to a known object, experience can be used for grasp synthesis [5, 6, 7, 8]. An unknown object needs to be analyzed in terms of its 3D structure and other physical properties from which a suitable grasp can be inferred [9, 10, 11, 12].

Realistic applications require going beyond open-loop execution of these grasps and the ability to deal with different type of errors occurring in an integrated robotic system. One kind of errors are systematic and repeatable, introduced mainly by inaccurate kinematic models. These can be minimized offline through precise calibration. The second kind are random errors introduced by a limited repeatability of the motors or sensor noise. These have

to be compensated through online mechanisms.

In this paper, we show how *Visual Servoing* (VS) can be used at different stages in a grasping pipeline to correct errors both offline and online. One of the contributions is the application of VS for automatic calibration of the hardware. We follow the classical approach of applying VS for aligning the robot hand with the object prior to grasping it [13]. This requires tracking of the manipulator pose relative to the camera. Instead of the common approach of putting markers on the robot hand, we use a model based tracking system. This is achieved through *Virtual Visual Servoing* (VVS) in which the systematic and random errors are compensated for. An additional contribution is the generalisation of VVS to CAD models instead of using object models tailor-made for the application.

The remainder of this paper is organised as follows. Section 2 formalises the systematic and random errors inherent to the different parts of the robotic system. In Section 3, related work in the area of offline calibration as well as closed-loop control is presented. The approach proposed in this paper is described in detail in Section 4. Its performance is analysed quantitatively on synthetic data and qualitatively on our robotic platform. The results of these experiments are presented in Section 5.

2. Problem Formulation

Object grasping in real world scenarios requires a set of steps to be performed prior to the actual manipulation of an object. A general outline of our grasping pipeline is provided in Figure 2. The hardware components of the system as shown in Figure 1 are (i) the Armar III robotic head [14] equipped with two stereo camera pairs (wide-angle for peripheral vision and narrow-angle for foveal vision), (ii) the 6 DoF Kuka arm KR5 sixx R850 [15] and (iii) the Schunk Dexterous Hand 2.0 (SDH) [16].

2.1. Grasping Pipeline

The main pre-requisite for a robot to perform a pick-and-place task is to have an understanding of the 3D environment it is acting in. In our pipeline, we perform scene construction by using the active head for visual exploration and stereo reconstruction as described in detail in our previous work [17]. The resulting point cloud is segmented into object hypotheses and background.

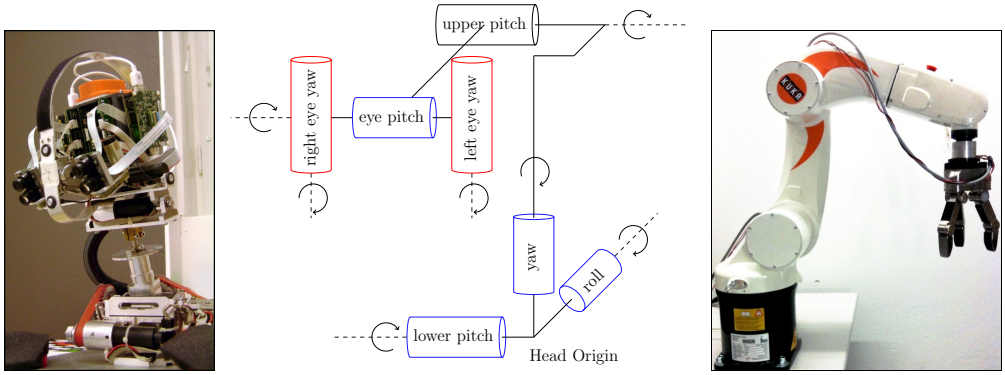


Figure 1: Hardware Components in the Grasping Pipeline. 1(a) Armar III Active Head with 7 DoF. 1(b) The Kinematic Chain of the Active Head. The upper pitch is kept static. Right and left eye yaw are actuated during fixation and thereby change the vergence angle and the epipolar geometry. All the other joints are used for gaze shifts. 1(c) Kuka Arm with 6 DoF and Schunk Dexterous Hand 2.0 (SDH) with 7 DoF.

The scene model then consists of the arm and the active head positioned relative to each other based on the offline calibration as described in Section 4.4. Furthermore, a table plane is detected and the online detected object hypotheses are placed on it.

Grasp inference is then performed on each hypothesis. For given grasp candidates, a collision-free arm trajectory is planned in the scene model and applied to the object hypothesis with the real arm.

2.2. Error Formalisation

The aforementioned grasping pipeline relies on the assumption that all the parameters of the system are perfectly known. This includes e.g. internal and external camera parameters as well as the pose of the head, arm and hand in a globally consistent coordinate frame.

In reality however two different types of errors are inherent to the system. One contains the systematic errors that are repeatable and arise from inaccurate calibration or inaccurate kinematic models. The other group comprises random errors originating from noise in the motor encoders or in the camera. These errors propagate and deteriorate the relative alignment between hand and object. The most reliable component of the system is the Kuka arm that has a repeatability of less than 0.03 mm [15]. In the following, we will analyse the different error sources in more detail.

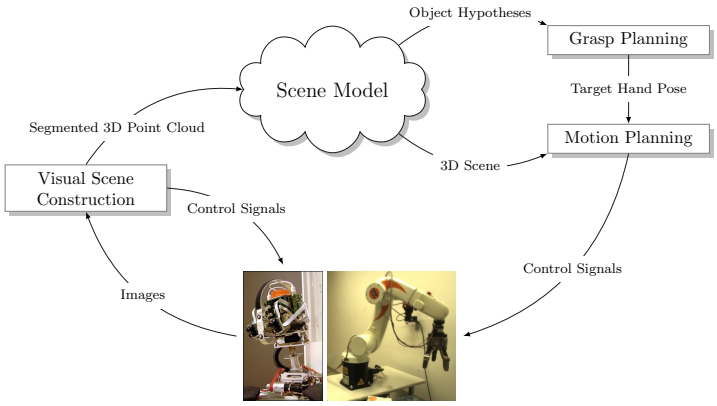


Figure 2: Our Open-Loop Grasping Pipeline.

2.2.1. Stereo Calibration Error

Given a set of 3D points $Q^W = \begin{bmatrix} x^W & y^W & z^W \end{bmatrix}^{\mathrm{T}}$ in the world reference frame W and their corresponding pixel coordinates $P = \begin{bmatrix} u & v & 1 \end{bmatrix}^{\mathrm{T}}$ in the image, we can determine the internal parameters $\mathbf{C}$ and external parameters $[\mathbf{R}_W^C|\mathbf{t}_W^C]$ for all cameras C in the vision system. This is done through standard methods by exploiting the following relationship between Q^W and P:

$$wP = \begin{bmatrix} wu \\ wv \\ w \end{bmatrix} = \mathbf{C}P^C \text{ with } P^C = \begin{bmatrix} z^C \\ y^C \\ z^C \end{bmatrix} = \mathbf{R}_W^C \left[P^W - \mathbf{t}_W^C \right] \tag{1}$$

Once we have these parameters for the left camera L and right camera R, the epipolar geometry as defined by the essential matrix $\mathbf{E} = \mathbf{R}_L^R[\mathbf{t}_L^R]_\times$ can be determined. Here $\mathbf{t}_L^R$ defines the baseline and $\mathbf{R}_L^R$ the rotation between the left and right camera system.

Our calibration method, which uses the arm as world reference frame, will be described in Section 4.4. The average reprojection error is 0.1 pixels. Since peripheral and foveal cameras are calibrated simultaneously, we also get the transformation between them.

Additionally to the systematic error in the camera parameters, other effects such as camera noise and specularities lead to random error in the stereo matching.

2.2.2. Positioning Error of the Active Head

We are using the robot head to actively explore the environment through gaze shifts and fixation on objects. This involves dynamically changing the epipolar geometry between the left and right camera system. Also the camera position relative to the head origin is changed. The internal parameters remain static.

The kinematic chain of the active head is shown in Figure 1(b). Only the last two joints, the left and right eye yaw, are used for fixation and thereby affect the stereo calibration. The remaining joints are actuated for performing gaze shifts.

In order to accurately detect objects with respect to the camera, the relation between the two camera systems as well as between the cameras and the head origin needs to be determined after each movement. Ideally, these transformations should be obtainable just from the known kinematic chain of the robot and the readings from the encoders. In reality, these readings are erroneous due to noise and inaccuracies in the kinematic model.

Let us first consider the epipolar geometry and assume that the left camera system defines the origin and remains static. Only the right camera rotates around its joint by ϕ at time k. According to the kinematic model this movement can be expressed by

$$[\mathbf{R}^k_{k-1}|\mathbf{t}^k_{k-1}] \text{ with } \mathbf{R}^k_{k-1} = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } \mathbf{t}^k_{k-1} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^{\mathrm{T}} \quad (2)$$

which is a pure rotation around the z-axis of the joint. The new essential matrix would then be

$$\mathbf{E}_k = \mathbf{R}_k \mathbf{R}^R_L [\mathbf{t}^R_L]_\times. \quad (3)$$

Inaccuracies arise in the manufacturing process influencing the true center and axis of joint rotations and in the discrepancy between motor encoder readings and actual angular joint movement. This is illustrated in Figure 3 showing the translational component $\delta = \begin{bmatrix} \delta_x & \delta_y & \delta_z \end{bmatrix}^{\mathrm{T}}$ and rotational component $\epsilon = \begin{bmatrix} \epsilon_x & \epsilon_y & \epsilon_z \end{bmatrix}^{\mathrm{T}}$ of the error between the ideal and real joint positions. Under the assumption that only small angle errors occur, the error matrix can be approximated as [18]

$$[\mathbf{R}_e|\mathbf{t}_e] \text{ with } \mathbf{R}_e = \begin{bmatrix} 1 & -\epsilon_z & \epsilon_y \\ \epsilon_z & 1 & -\epsilon_x \\ -\epsilon_y & \epsilon_x & 1 \end{bmatrix} \text{ and } \mathbf{t}_e = \begin{bmatrix} \delta_x & \delta_y & \delta_z \end{bmatrix}^{\mathrm{T}} \quad (4)$$

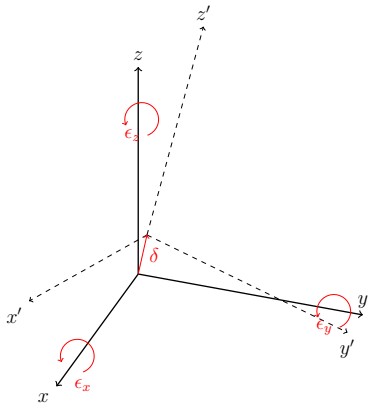


Figure 3: Six parametric errors in a rotary joint around the z-Axis. $\delta = [\delta_x\ \delta_y\ \delta_z]^{\mathrm{T}}$ is the translational component and $\epsilon = [\epsilon_x\ \epsilon_y\ \epsilon_z]^{\mathrm{T}}$ is the rotational component of the error.

Based on this, the true essential matrix can be modelled as

$$\mathbf{E}_k = \mathbf{R}_e \mathbf{R}_k \mathbf{R}_L^R [\mathbf{R}_e \mathbf{t}_L^R + \mathbf{t}_e]_\times. \tag{5}$$

Leaving this specific error matrix unmodelled will lead to erroneous depth measurements of the scene.

Similarly to the vergence angle, error matrices can be defined for every joint modelling inaccurate positioning and motion. Let us define J_{n-1} and J_n as two subsequent joints. According to the Denavit-Hartenberg convention, the ideal transformation $\mathbf{T}_{n-1}^n$ between these joints is defined as

$$\mathbf{T}_{n-1}^n = {}_z\mathbf{T}_{n-1}^n(d, \phi)\ \ {}_x\mathbf{T}_{n-1}^n(a, \alpha) \tag{6}$$

where ${}_z\mathbf{T}_{n-1}^n(d, \phi)$ describes the translation d and rotation ϕ with respect to the z-axis of J_{n-1}. ${}_x\mathbf{T}_{n-1}^n(a, \alpha)$ describes the translation a and rotation α with respect to the x-axis of J_n. While d, a and α are defined by the kinematic model of the head, ϕ is varying with the motion of the joints and can be read from the motor encoders.

The true transformation ${}_e\mathbf{T}_{n-1}^n$ will however look different. Modelling the position error $\mathbf{T}_{pe}$ and motion error $\mathbf{T}_{me}$ as in Equation 4 yields

$${}_e\mathbf{T}_{n-1}^n = {}_z\mathbf{T}_{n-1}^n(d, \phi)\mathbf{T}_{me}\ \ {}_x\mathbf{T}_{n-1}^n(a, \alpha)\mathbf{T}_{pe}. \tag{7}$$

These errors propagate through the kinematic chain and mainly affect the x and y position of points relative to the world coordinate system.

Regarding random error, the last five joints in the kinematic chain achieve a repeatability in the range of ± 0.025 degrees [14]. The neck pitch and neck roll joints in Figure 1(b) achieve a repeatability in the range of ± 0.13 and ± 0.075 degrees respectively.

2.2.3. Positioning Error of the Cameras with Respect to the Arm

As will be described in Section 4.4, we are using the arm to calibrate the stereo system. Therefore, assuming that the transformation from the head origin to the cameras is given, the error in the transformation between the arm and cameras is equivalent to the error of the stereo calibration.

3. Related Work

3.1. Closed-Loop Control in Robotic Grasping

In the previous section, we summarised the errors in a grasping system that lead to an erroneous alignment of the end effector with an object. A system that executes a grasp in closed loop without any sensory feedback is likely to fail.

In [19, 20] this problem is tackled by introducing haptic and force feedback into the system. Control laws are defined that adapt the pose of the end effector based on the readings from a force-torque sensor and contact location on the haptic sensors. A disadvantage of this approach is that the previously detected object pose might change during the alignment process.

Other grasping systems make use of visual feedback to correct the wrong alignment before contact between the manipulator and the object is established. Examples are proposed by Huebner et al. [2] and Ude et al. [21], who are using a similar robotic platform to ours including an active head. In [2], the Armar III humanoid robot is enabled to grasp and manipulate known objects in a kitchen environment. Similar to our system [17], a number of perceptual modules are at play to fulfill this task. Attention is used for scene search. Objects are recognized and their pose estimated with the approach originally proposed in [4]. Once the object identity is known, a suitable grasp configuration can be selected from an offline constructed database. Visual servoing based on a wrist spherical marker is applied to bring the robotic hand to the desired grasp position [22]. Different from our approach, absolute 3D data is estimated by fixing the 3 DoF for the eyes to a position for which a stereo calibration exists. The remaining degrees of freedom controlling the neck of the head are used to keep the target and current hand

position in view. In our approach, we reconstruct the 3D scene by keeping the eyes of the robot in constant fixation on the current object of interest. This ensures that the left and right visual field overlap as much as possible, thereby maximizing e.g. the amount of 3D data that can be reconstructed. However, the calibration process becomes much more complex.

In the work by Ude et al. [21], fixation plays an integral part of the vision system. Their goal is however somewhat different from ours. Given that an object has been already placed in the hand of the robot, it moves it in front of its eyes through closed loop vision based control. By doing this, it gathers several views from the currently unknown object for extracting a view-based representation that is suitable for recognizing it later on. Different to our work, no absolute 3D information is extracted for the purpose of object representation. Furthermore, the problem of aligning the robotic hand with the object is circumvented.

3.2. Calibration Methods

In [23], the authors presented a method for calibrating the active stereo head. The correct depth estimation of the system was demonstrated by letting it grasp an object held in front of its eyes. No dense stereo reconstruction has been shown in this work.

Similarly, in [24] a procedure for calibrating the Armar III robotic head was presented. Our calibration procedure is similar to the one described in those papers, with a few differences. We extend it to the calibration of all joints, thus obtaining the whole kinematic chain. Also, the basic calibration method is modified to use an active pattern instead of a fixed checkerboard, which has some advantages that we outline in Section 4.4.

3.3. Visual and Virtual Visual Servoing

For the accurate control of the robotic manipulator using visual servoing, it is necessary to know its position and orientation (*pose*) with respect to the camera. In the systems mentioned in Section 3.1, the end effectors of the robots are tracked based on fiducial markers like LEDs, colored spheres or Augmented Reality tags. A disadvantage of this marker based approach is that the mobility of the robot arm is constrained to keep the marker always in view. Furthermore, the position of the marker with respect to the end effector has to be known exactly. For these reasons, we propose to track the whole manipulator instead of only a marker. Thereby, we alleviate the problem of constrained arm movement. Additionally, collisions with the

object or other obstacles can be avoided in a more precise way. In this paper we track the pose of the robotic arm and hand assuming their kinematic chain to be perfectly calibrated, although the system can be adapted to estimate deviations in the kinematic chain.

Tracking objects of complex geometry is not a new problem and approaches can be divided into two groups: appearance-based and model-based methods. The first approach is based on comparing camera images with a huge database of stored images with annotated poses [25]. The second approach relies on the use of a geometrical model (3D CAD model) of the object and perform tracking based on optical flow [26]. There have also been examples that integrate both of these approaches [27].

Apart from the tracking itself, an important problem is the initialization of the tracking process. This can be done by first localizing the object in the image followed by a global pose estimation step. In our previous work, we have also demonstrated how the initialization can be done for objects in a generic way [28]. In the case of a manipulator, a rough estimate of its pose in the camera frame can be obtained from the kinematics of the arm and the hand-eye calibration. In [29], it has been shown that the error between this first estimate and the real pose of the manipulator can be corrected through virtual visual servoing, using a simple wireframe model of the object. In our work, a synthetic image of the robot arm is rendered based on a complete 3D CAD model and its initial pose estimate is compared and aligned with the real image.

In our recent work [30], we demonstrate how pose estimation can be performed for a complex articulated object such as a human hand. The major contribution of that work is the use of a discriminative machine learning approach for obtaining real-time tracking of an object with 27 degrees of freedom. The problem considered in this paper has a lower dimensionality and a more accurate guess of the initial pose. This makes it possible to adopt a real-time generative approach that renders the last pose of the object in each frame and estimates the new pose through a process of error minimization.

4. The Proposed System

For overcoming the problem of lacking a globally consistent coordinate frame for objects, manipulator and cameras, we introduce visual servoing into the grasping pipeline. This allows us to control the manipulator in closed

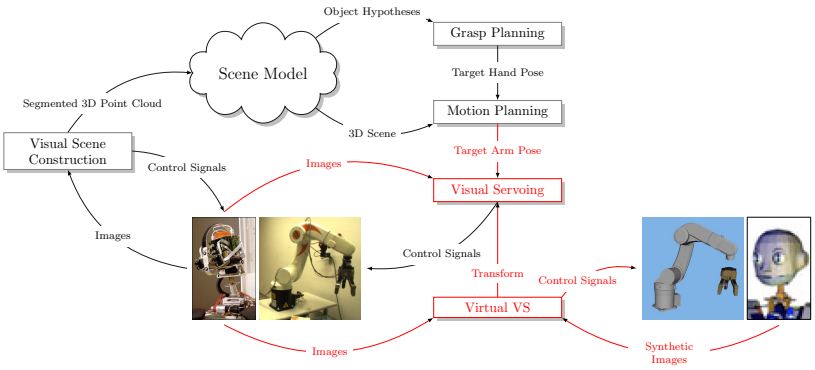


Figure 4: A Grasping Pipeline With Closed Loop Control of the Manipulator through Visual Servoing that is initialised through Virtual Visual Servoing. Armar III Head Model adapted from [31].

loop using visual feedback to correct any misalignment with the object online. We then use the camera coordinate frame as the global reference system in which the manipulated object and robot are also defined.

For accurately tracking the pose of the manipulator, we use virtual visual servoing. The initialisation of this method is based on the known joint values of hand and arm and the hand-eye calibration, which is obtained offline.

The adapted grasping pipeline is summarised in Figure 4. What follows is a more detailed description of all the components of this pipeline.

4.1. Vision System for Constructing the Scene Model

As described earlier, the scene model in which grasping is performed consists of a robot arm, hand and head as well as a table plane onto which object hypotheses are placed. The emergence of these hypotheses is triggered by the visual exploration of the scene with the robotic head. In the following, we will give a brief summary of this exploration process. More details can be found in our previous work [3, 32, 33, 17].

The active robot head has two stereo camera pairs. The wide-field cameras, of which an example can be seen in Figure 5(a), are used for scene search. This is done by computing a saliency map on them and assuming that maxima in this map are initial object hypotheses (Figure 5(b)). A gaze shift is performed to a maxima such that the stereo camera with the narrow-angle lenses center on the potential object. An example of an object fixated in the foveal view is shown in Figure 5(c). In the following we will label the

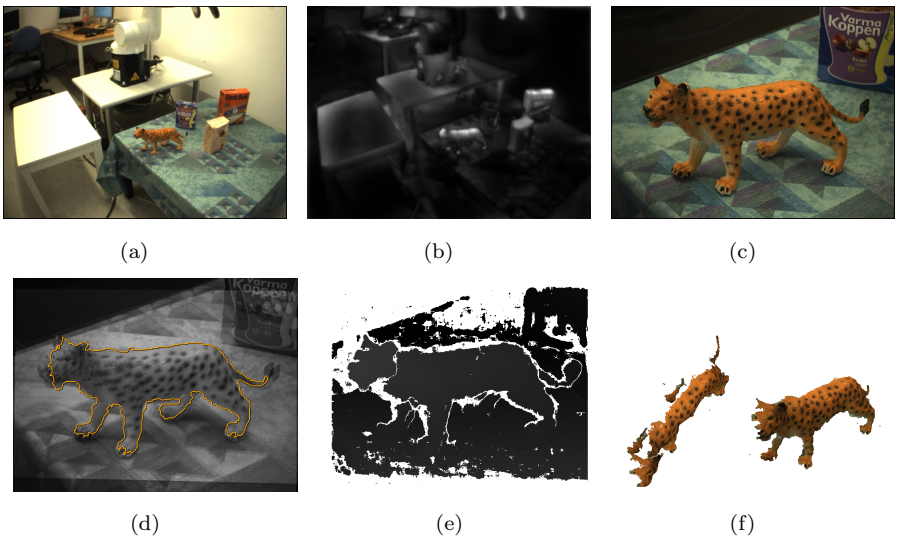


Figure 5: Example Output for Exploration Process. 5(a) Left Peripheral Camera. 5(b) Saliency Map of Peripheral Image. 5(c) Left Foveal Camera. 5(d) Segmentation on Overlayed Fixated Rectified Left and Right Images. 5(e) Disparity Map. 5(f) Point Cloud of Tiger.

camera pose when fixated on the current object of interest as C_0. Once the system is in fixation, a disparity map is calculated and segmentation performed (see Figure 5(e) and 5(d)). For each object, we then obtain a 3D point cloud (an example is shown in Figure 5(f)).

4.2. Grasp and Motion Planning

Once a scene model has been obtained by the vision system, a suitable grasp can be selected for each object hypothesis depending on the available knowledge about the object. In our previous work, we presented grasp planners on known, familiar or unknown objects [2, 5, 31, 17]. In [2, 31] resulting grasp hypotheses are tested in simulation on force closure prior to execution. The set of stable grasps is then returned to plan corresponding collision free arm trajectories.

The focus of this paper is the application of visual servoing and virtual visual servoing in the grasping pipeline. We have therefore selected a simple top-grasp selection mechanism that has been proven to be very effective for pick-and-place tasks in table-top scenarios [17]. For this, we calculate the eigenvectors and centre of mass of the projection of the point cloud on the

table plane. An example of such a projection can be seen in Figure 5(f) (left). The corresponding grasp is a top grasp approaching the centre of mass of this projection. The wrist orientation of the hand is determined such that the vector between fingers and thumb is aligned with the minor eigenvector. A grasping point $G^{C_0} = \begin{bmatrix} zx & zy & z \end{bmatrix}$ in camera space C_0 is then formed with the x and y coordinates of the center of the segmentation mask as obtained during fixating on the object. The depth of this point, i.e the z coordinate, is computed from the vergence angle as read from the motor encoders. The goal is then to align the tool center point of the end effector with this grasping point. Using more sophisticated grasp planners that can deal with more complex scenarios is regarded as future work.

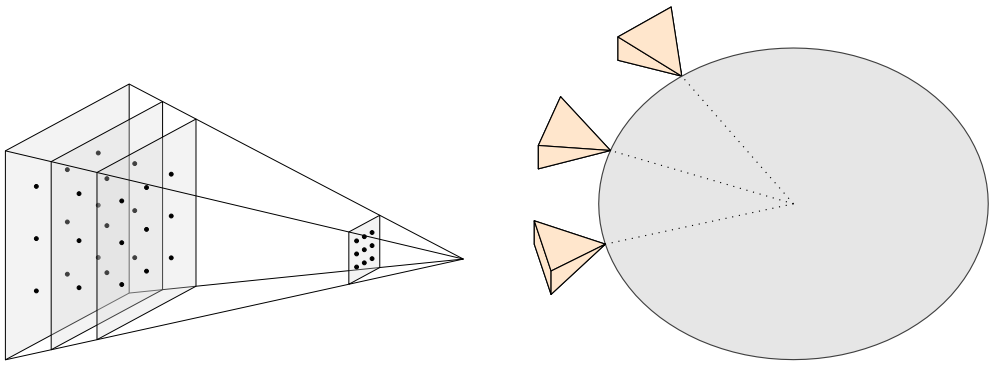
4.3. Object Localisation in Different Viewing Frames

The robotic head moves during the grasping process for focusing on different parts of the environment (different objects, the robotic hand and arm). In each of these views the object to be grasped should remain localized relative to the current camera frame C_k to accurately align the end effector with it. The new position of the object can be estimated given the new pose of the head, which is determined based on the motor reading and the forward kinematics of the head as depicted in Figure 1(b). However, we cannot completely rely on this estimate due to inaccuracies in the kinematic model and motor encoders. For this reason we perform a local refinement of the object position in the new view of the scene based on template matching.

A template is generated for each object when the head is fixated on it. Based on the segmentation mask in the foveal view as shown in Figure 5(d) and the calibration between the peripheral and foveal cameras, we generate a tight bounding box around the object in the peripheral view. Each time the head moves, the new position of this template can be estimated based on the old and new pose of the head. We create a window of possible positions of the object by growing this initial estimate by a fixed amount of pixels. We perform a sliding window comparison between all possible locations of the object template within this window and the object template. The location which returns the lowest mean square error is the one suggested as x and y coordinates of the object position in the new view.

4.4. Offline Calibration Process

In Section 2.2, we analysed the errors that got introduced by the different parts of the system. The group of systematic errors can be minimised by



(a) Movement pattern of the end effector for offline calibration.

(b) Three viewing directions of a camera rotated around one axis.

Figure 6: Offline Calibration procedure (video available at `http://www.youtube.com/watch?v=dEytfUgmcfA`).

offline calibration. In the following, we will introduce our method for stereo and hand-eye calibration as well as the calibration of the kinematic chain of the robotic head.

4.4.1. Stereo Calibration

One of the most commonly used methods for finding the transformation between two camera coordinate systems is the use of a checkerboard which is observed by two cameras (or the same camera before and after moving) [34]. The checkerboard defines its own coordinate system in which the corners of the squares are the set of 3D points Q^A in the arm coordinate system A as in Section 2.2.1. The detection of these corners in the left and right images gives us the corresponding pixel coordinates P from which we can solve for the internal and external camera parameters. Given these, we can obtain the transformation between the left and right camera coordinate frame.

We used a modified version of this method. Instead of a checkerboard pattern, we used a small LED rigidly attached to the end effector of the robotic arm, which we can detect in the image with subpixel precision. Because of the accuracy and repeatability of the KUKA arm (< 0.3mm), we can move the LED to a number of places for which we know the exact position in arm space, which allows us to obtain the transformation between arm and camera space.

This method has several advantages over the use of a traditional checkerboard pattern:

- Instead of an arbitrary checkerboard coordinate system as intermediate coordinate system, we can use the arm coordinate system. In this way we are obtaining the hand-eye calibration for free at the same time that we are performing the stereo calibration.

- In the checkerboard calibration method, the checkerboard ought to be fully visible in the two calibrating cameras for every calibration pose. This may be difficult when the two camera poses are not similar or their field of view is small. With our approach, it is not necessary to use exactly the same end effector positions for calibrating the two cameras since all points are defined in the static arm coordinate system that is always valid independently of camera pose.

- For these same reasons, we found empirically that this approach makes it possible to choose a pattern that offers a better calibration performance. For example, by using a set of calibration points that is uniformly distributed in camera space (as opposed to world space, which is the case for checkerboard patterns), it is possible to obtain a better characterization of the lenses distortion parameters.

The main building block for this system is a visual servoing loop that allows us to bring the LED to several predefined positions in camera space. In this loop, we want to control the position of the LED (3 DOF) using only its projection in the image (2 DOF). Therefore, we introduce an additional constraint by limiting the movement of the LED to a predefined plane in arm space.

We define S_0^A as a point on this plane and S^A as its normal vector. Additionally, we need a rough estimate of the arm to camera coordinate transformation $\boldsymbol{A}_A^C$, and of the camera matrix $\boldsymbol{C}$. They are defined as follows:

$$\boldsymbol{A}_A^C = \left[\boldsymbol{R}_A^C | \boldsymbol{t}_A^C\right], \boldsymbol{C} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

We can then find the plane in camera space:

$$S_0^C = \boldsymbol{R}_A^C S_0^A + \boldsymbol{t}_A^C \qquad S^C = \boldsymbol{R}_A^C S^A \tag{9}$$

The process of moving the LED to a position in the image is then as follows: we define a target point in the image $P_t = (u_t, v_t)$ (specified in

pixels) where we want to bring the LED. For each visual servoing iteration we detect the current position $P_c = (u_c, u_c)$ (again in pixels) of the LED in the image. We can then use the camera matrix to convert these points to homogeneous camera coordinates:

$$P_t^C = \boldsymbol{C}^{-1} \begin{bmatrix} x_t & y_t & 1 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} \frac{1}{f} x_t & \frac{1}{f} y_t & 1 \end{bmatrix}^{\mathrm{T}} \tag{10}$$

$$P_c^C = \boldsymbol{C}^{-1} \begin{bmatrix} x_c & y_c & 1 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} \frac{1}{f} x_c & \frac{1}{f} y_c & 1 \end{bmatrix}^{\mathrm{T}} \tag{11}$$

Then, we can project these homogeneous points into the plane defined by S_0^C and S^C. A point Q is in that plane if $Q \cdot S^C = S_0^C \cdot S^C$. Therefore, the projection of the homogeneous points P_t^C and P_c^C into the plane can be found as

$$Q_t^C = \frac{S_0^C \cdot S^C}{P_t^C \cdot S^C} P_t^C \qquad Q_c^C = \frac{S_0^C \cdot S^C}{P_c^C \cdot S^C} P_c^C \tag{12}$$

From these, we can obtain the vector

$$d^C = Q_t^C - Q_c^C \tag{13}$$

which is the displacement from the current to the target LED positions in camera space, and then

$$d^A = \boldsymbol{R}_A^{C^{-1}} d^C \tag{14}$$

which is the correspondent displacement in arm space.

We can easily see that the displacement obtained in arm space is within the given plane since

$$d^C \cdot S^C = (Q_t^C - Q_c^C) \cdot S^C = Q_t^C \cdot S^C - Q_c^C \cdot S^C = 0 \tag{15}$$

$$d^A \cdot S^A = d^{A\mathrm{T}} S^A = (\mathbf{R}^{-1} d^C)^{\mathrm{T}} (\mathbf{R}^{-1} S^C) = d^{C\mathrm{T}} \mathbf{R} \mathbf{R}^{-1} s^C = d^C \cdot S^C = 0 \tag{16}$$

We can then use this displacement to obtain a simple proportional control law

$$\boldsymbol{u} = k d^A \tag{17}$$

where $\boldsymbol{u}$ is the velocity of the end effector and k is a constant gain factor.

Once we have the system which allows us to bring the LED to a certain position in the image we can start generating our calibration pattern. First, we bring the LED to the center of the image in two parallel planes, which are located at different distances from the camera, and record their positions C_0^A and C_1^A in arm space. From this, we can obtain the vector $S^A = C_0^A - C_1^A$

Algorithm 1: Pseudo Code for Defining the Calibration Pattern

Data: Number of points n on each depth plane, number of depth planes m
Result: Set of Correspondences $\left[Q_i^A, P_i\right]$ with $(0 < i \leq m \cdot n)$
begin
  `// Initialising the principal axis of the camera in arm space`
  `// VisualServoing(P,S) is a function that brings the LED to`
  `// the point P in image space within the plane S in arm space`
  $\mathbf{S}_0$ = Some plane at a distance d_0 from the camera
  C_0^A = `VisualServoing`$((0,0), \mathbf{S}_0)$
  $\mathbf{S}_1$ = Some plane at a distance d_1 from the camera
  C_1^A = `VisualServoing`$((0,0), \mathbf{S}_1)$
  $d = (C_1^A - C_0^A)/(m-1)$
  $i = 0$
  foreach $l \in [0 \ldots m-1]$ **do**
    $\mathbf{S}$ = plane with normal d which contains $C_0^A + ld$
    foreach P_k *with* $(0 < k \leq n)$ **do**
      Q_i^A = `VisualServoing`$((x_k, x_y), \mathbf{S})$
      $i++$
    end
  end
end

which is parallel to the principal axis of the camera. Any plane perpendicular to this vector will be parallel to the image plane. We can then bring the LED to some fixed positions along these planes, thus generating points which are uniformly distributed both in the image and in depth (by using planes which are separated by a constant distance). The generated pattern looks like the one shown in Figure 6(a). Algorithm 1 shows the method in more detail. In our system, we used 6x6 points rectangular patterns in 6 different depths, for a total of 216 calibration points. With this, we achieved an average reprojection error of 0.1 pixels.

4.4.2. Head-Eye Calibration

For a static camera setup, the calibration process would be completed here. However, our vision system can move to fixate on the objects we manipulate. Due to inaccuracies in the kinematic model of the head, we cannot obtain the exact transformation between the camera coordinate system before and after moving a certain joint from the motor encoders.

In Section 2.2.2, we modelled the transformation error that arises from the misalignment of the real and ideal coordinate frame of a joint. In our method,

we are minimising this error by finding the true position and orientation of the real coordinate frame as follows:

1. Choose two different positions of the joint, that are far enough apart to be significant, but with an overlapping viewing area that is still reachable for the robotic arm.
2. For each of these two positions, perform the static calibration process as described above, so that we obtain the transformation between the arm coordinate system and each of the camera coordinate systems.
3. Find the transformation between the camera coordinate systems in the two previously chosen joint configurations. This transformation is the result of rotating the joint around some roughly known axis (it is not precisely known because of mechanical inaccuracies), with a roughly known angle from the motor encoders. From the computed transformation, we can then more exactly determine this axis, center and angle of rotation.

This is illustrated in Figure 6(b).

In our case, the results showed that while the magnitude of the angles of rotation differed significantly from what could be obtained from the kinematic chain, the orientation and position of these axes were quite precise in the specifications. To avoid overly complicating the model, we decided to correct only the actual angles α in Equation 6 , except for the vergence joint. For this joint, the orientation was also corrected, since here small errors have a large impact in the accuracy of depth estimation. Therefore, we minimised the discrepancy between the true and estimated essential matrix in Equations 3 and 5 respectively.

4.5. Virtual Visual Servoing

As mentioned in Sections 1 and 3.3, our system uses Virtual Visual Servoing to refine the position of the arm and hand in camera space provided by the calibration system. In Figure 8, the difference between the estimated arm pose and the corrected one is visualised. In this section we will formalize the problem and explain how we solved it.

The pose of an object is denoted by $\mathbf{M}(\mathbf{R}, \mathbf{t})$ where $\mathbf{t} \in \mathcal{R}(3)$, $\mathbf{R} \in \mathbf{SO}(3)$. The set of all poses that the robot's end–effector can attain is denoted with $\mathcal{T}_G \subseteq \mathbf{SE}(3) = \mathcal{R}(3) \times \mathbf{SO}(3)$.

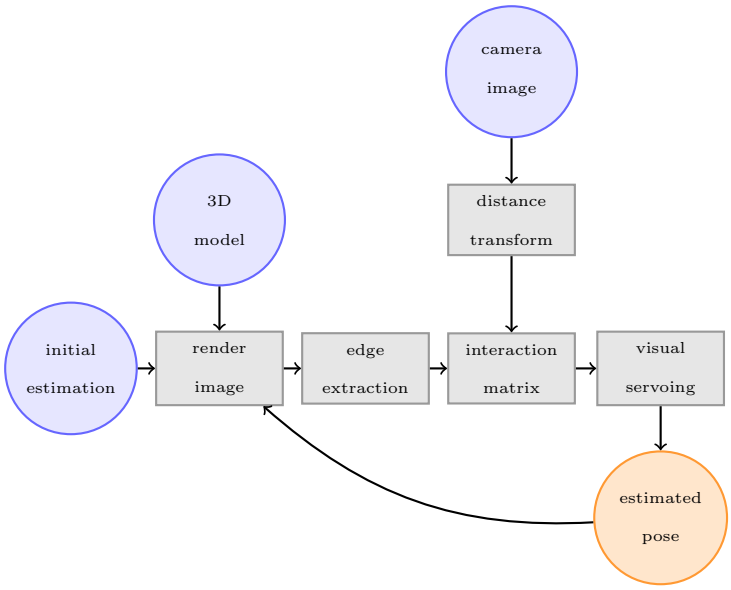
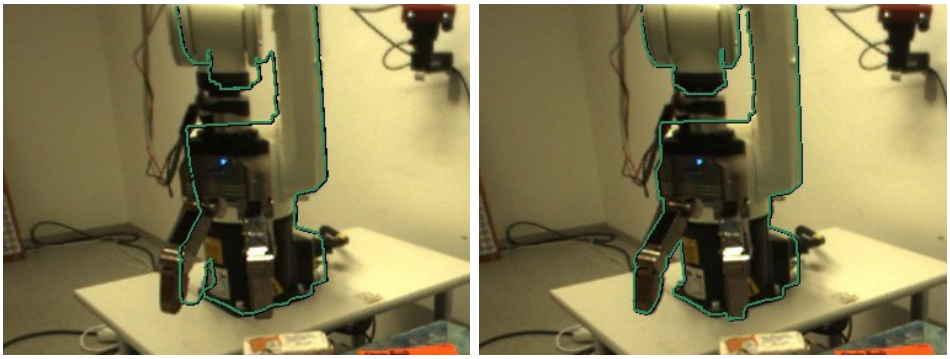


Figure 7: Outline of the proposed model-based tracking system based on Virtual Visual Servoing.



(a) (b)

Figure 8: Comparison of robot localization with (right) and without (left) the Virtual Visual Servoing correction.

Pose estimation considers the computation of a rotation matrix (orientation) and a translation vector of the object (position), $\mathbf{M}(\mathbf{R}, \mathbf{t})$:

$$\mathbf{M} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & T_X \\ r_{21} & r_{22} & r_{23} & T_Y \\ r_{31} & r_{32} & r_{33} & T_Z \end{bmatrix} \tag{18}$$

The equations used to describe the projection of a three–dimensional model point $\mathbf{Q}$ into homogeneous coordinates of the image point $[x\ y]^{\mathrm{T}}$ are:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \mathbf{R}\left[\mathbf{Q} - \mathbf{t}\right] \quad \text{with} \quad [wx,\ wy,\ w] = \mathbf{P} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{19}$$

where $\mathbf{P}$ represents the internal camera parameters matrix including focal length and aspect ratio of the pixels, w is the scaling factor, $\mathbf{R}$ and $\mathbf{t}$ represent the rotation matrix and translation vector, respectively.

Our approach to pose estimation and tracking is based on virtual visual servoing where a rendered model of the robot parts is aligned with the current image of their real counterparts. The outline of the system is presented in Fig. 7. In order to achieve the alignment, we can either control the position of the part to bring it to the desired pose or move the *virtual* camera so that the *virtual* image corresponds to the current camera image, denoted as *real* camera image in the rest of the paper. Using the latter approach has the problem that the local effect of a small change in orientation of the camera is very similar to a large change in its position, which leads to convergence problems. Therefore, in this paper, we adopt the first approach where we render synthetic images by incrementally changing the pose of the tracked part. In the first iteration, the position is given based on the forward kinematics. Then, we extract visual features from the rendered image, and compare them with the features extracted from the current camera image. The details about the features that are extracted are given in Section 4.5.2.

Based on the extracted features, we define an error vector

$$\boldsymbol{s}(t) = [d_1, d_2, \ldots, d_n]^{\mathrm{T}} \tag{20}$$

between the desired values for the features and the current ones. Based on $\boldsymbol{s}$, we can estimate the incremental change in pose in each iteration $\boldsymbol{e}(s)$ following the classical image based servoing control loop. This process continues

until the difference vector $\boldsymbol{s}$ is smaller than a certain threshold. Each of the steps is explained in more detail in the following subsections. Our current implementation and experimental evaluation is performed for the Kuka R850 arm and Schunk Dexterous hand, shown in Fig. 9

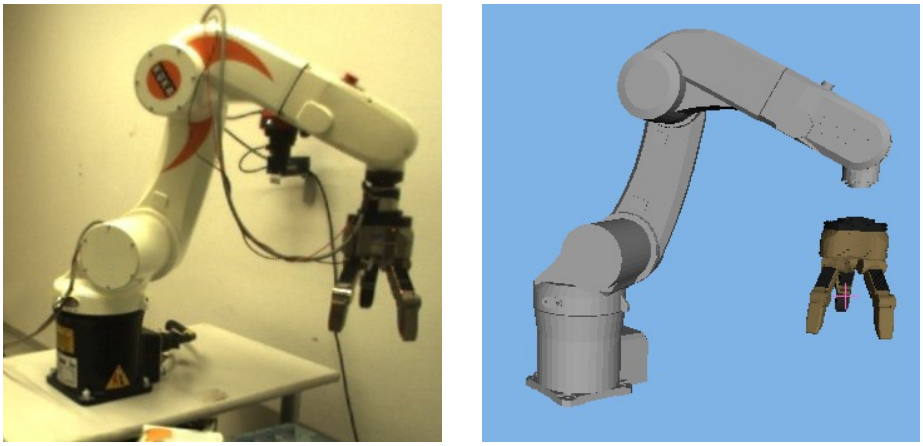


Figure 9: The Kuka R850 arm and Schunk Dexterous Hand in real images and CAD models.

4.5.1. Virtual image generation

As we mentioned before, we use a realistic 3D model of the robotic parts as input for our system. This adds the challenge of having to render this model at a high frame rate, since our system runs in real time, and several visual servoing iterations must be performed for every frame that we obtain from the cameras.

To render the image, we use a projection matrix $\boldsymbol{P}$, which corresponds to the internal parameters of the real camera, and a modelview matrix $\boldsymbol{M}$, which consists of a rotation matrix and a translation vector. The modelview matrix is then estimated in the visual servoing loop.

One of the most common CAD formats for objects such as robotic hands are Inventor files. There are a number of rendering engines which can deal with such models, but none of them had the performance and flexibility that we needed. For that we developed a new scenegraph engine, specific for this application, which focused on rendering offline images at a high speed. It was developed directly over OpenGL. We can obtain about 1000 frames per second with this engine. At the moment the speed of the rendering engine does not represent a bottleneck to our system.

We render the image without any texture or lighting, since we are only interested in the external edges of the model. We also save the depth map produced by the rendering process, which will be useful later for the estimation of the jacobian.

4.5.2. Features

The virtual and the real image of the robot parts ought to be compared in terms of visual features. These features should be fast to compute, because this comparison will be performed as many times per frame as iterations are needed by the virtual visual servoing for locating the robot. They should also be robust towards non-textured models.

Edge-based features fulfill these requirements. In particular chamfer distances [35] modified to include the alignment of edge orientations [36] are well suited for matching shapes in cluttered scenes, and are used in recent systems for object localization [37]. This feature is similar in spirit but more general than other ones used in the field of Virtual Visual Servoing. Comport et al. [38] computes distances between real points and virtual lines/ellipsis instead of virtual points. Klose et al. [39], Drummond and Cipolla [40] search for real edges only in the perpendicular direction of the virtual edge.

The mathematic formulation of our features is the following. After performing a Canny edge detection on real image I_u and virtual image I_v we obtain a set of edge points $\mathcal{U}, \mathcal{V}$ with their correspondent edge orientations $\mathcal{O}_\mathcal{U}, \mathcal{O}_\mathcal{V}$ (from the horizontal and vertical Sobel filter). The sets $\mathcal{U}, \mathcal{V}$ are split into overlapping subsets $\mathcal{U}_i, \mathcal{V}_i$ according to their orientations:

$$o_u \pmod{\pi} \in [\frac{2\pi i}{16}, \frac{2\pi(i+1)}{16}] \Rightarrow u \in \mathcal{U}_i \tag{21}$$

Then for each channel $\mathcal{U}_i$ we compute the distance transform [41] which will allow us to perform multiple distance computations for the same real set $\mathcal{U}$ in linear time with the number of points in each new set $\mathcal{V}$. Based on the distance transform we obtain our final distance vector $\boldsymbol{s} = \{d_{cham}(v, \mathcal{U})\} = [d_1, d_2 \cdots d_n]^{\mathrm{T}}$ composed by distance estimations for each edge point from our virtual image I_v.

$$d_{transf}(p) = \min_{u \in \mathcal{U}_i} ||p - u||, \qquad p \in I_u \tag{22}$$

$$d_{cham}(v, \mathcal{U}) = d_{transf}(v), \qquad v \in \mathcal{V}_i, u \in \mathcal{U}_i \tag{23}$$

$$\boldsymbol{s} = \{d_{cham}(v, \mathcal{U})\}, \qquad d_{cham}(v, \mathcal{U}) < \delta \tag{24}$$

The points $v \in \mathcal{V}$ whose distance is higher than an empirically estimated threshold δ are considered outliers and dropped from $\boldsymbol{s}$.

4.5.3. Pose correction

Once the features have been extracted, we can use a classical visual servoing approach to calculate the correction to the pose, [42]. There are two different approaches to vision-based control: *Position-based* control uses image data to extract a series of 3D features, and control is performed in the 3D Cartesian space. In *image-based* control, the image features are directly used to control the robot motion. In our case, since the features we are using are distances between edges in an image for which we have no depth information in the real image, we are using an image-based approach.

The basic idea behind visual servoing is to create an error vector which is the difference between the desired and measured values for a series of features, and then map this error directly to robot motion.

As discussed before, $\boldsymbol{s}(t)$ is a vector of feature values measured in the image, composed by distances between edges in the real and synthetic images I_u, I_v. Therefore $\dot{\boldsymbol{s}}(t)$ will be the rate of change of these distances with time.

The movement of the manipulator (in this case, the virtual manipulator) can be described by a translational velocity $\boldsymbol{T}(t) = [T_x(t), T_y(t), T_z(t)]^{\mathrm{T}}$ and a rotational velocity $\boldsymbol{\Omega}(t) = [\omega_x(t), \omega_y(t), \omega_z(t)]^{\mathrm{T}}$. Together, they form a velocity screw:

$$\dot{\boldsymbol{r}}(t) = \left[T_x, T_y, T_z, \omega_x, \omega_y, \omega_z\right]^{\mathrm{T}} \tag{25}$$

We can then define the image jacobian or interaction at a certain instant as $\boldsymbol{J}$ so that:

$$\dot{\boldsymbol{s}} = \boldsymbol{J}\dot{\boldsymbol{r}} \tag{26}$$

where

$$\boldsymbol{J} = \left[\frac{\partial \boldsymbol{s}}{\partial \boldsymbol{r}}\right] = \begin{bmatrix} \frac{\partial d_1}{\partial T_x} & \frac{\partial d_1}{\partial T_y} & \frac{\partial d_1}{\partial T_z} & \frac{\partial d_1}{\partial \omega_x} & \frac{\partial d_1}{\partial \omega_y} & \frac{\partial d_1}{\partial \omega_z} \\ \frac{\partial d_2}{\partial T_x} & \frac{\partial d_2}{\partial T_y} & \frac{\partial d_2}{\partial T_z} & \frac{\partial d_2}{\partial \omega_x} & \frac{\partial d_2}{\partial \omega_y} & \frac{\partial d_2}{\partial \omega_z} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial d_n}{\partial T_x} & \frac{\partial d_n}{\partial T_y} & \frac{\partial d_n}{\partial T_z} & \frac{\partial d_n}{\partial \omega_x} & \frac{\partial d_n}{\partial \omega_y} & \frac{\partial d_n}{\partial \omega_z} \end{bmatrix} \tag{27}$$

which relates the motion of the (virtual) manipulator to the variation in the features. The method used to calculate the jacobian is described in detail

below.

However, to be able to correct our pose estimation, we need the opposite: we need to compute $\dot{\boldsymbol{r}}(t)$ given $\dot{\boldsymbol{s}}(t)$.

When $\boldsymbol{J}$ is square and nonsingular, it is invertible, and then $\dot{\boldsymbol{r}} = \boldsymbol{J}^{-1}\dot{\boldsymbol{s}}$. This is not generally the case, so we have to compute a least squares solution, which is given by

$$\dot{\boldsymbol{r}} = \boldsymbol{J}^{+}\dot{\boldsymbol{s}} \tag{28}$$

where $\boldsymbol{J}^{+}$ is the pseudoinverse of $\boldsymbol{J}$, which can be calculated as:

$$\boldsymbol{J}^{+} = (\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J})^{-1}\boldsymbol{J}^{\mathrm{T}}. \tag{29}$$

The goal for our task is to have all the edges in our synthetic image match edges in the real image, so the target value for each of the features is 0. Then, we can define the error function as

$$\boldsymbol{e}(\boldsymbol{s}) = \boldsymbol{0} - \dot{\boldsymbol{s}} \tag{30}$$

which leads us to the simple proportional control law:

$$\dot{\boldsymbol{r}} = -K\boldsymbol{J}^{+}\boldsymbol{s} \tag{31}$$

where K is the gain parameter.

4.5.4. Estimation of the jacobian

To estimate the jacobian, we need to calculate the partial derivatives of the feature values with respect to each of the motion components. When features are the position of points or lines, it is possible to find an analytical solution for the derivatives.

In our case, the features in $\boldsymbol{s}$ are the distances from the edges of the synthetic image to the closest edge in the real image. Therefore, we numerically approximate the derivative by calculating how a small change in the relevant direction affects the value of the feature.

As we said before, while rendering the model we obtained a depth map. From this depth map, it is possible to obtain the 3D point corresponding to each of the edges. Each model point v in I_v is a projection of its corresponding 3D point in the model $\boldsymbol{v_m}$. The derivative of the distance described before $d_{cham}(v, \mathcal{U})$, can be calculated for a model point $\boldsymbol{v_m}$ with respect to T_x by applying a small displacement and projecting it into the image:

$$\frac{d_{cham}\big(\boldsymbol{PM}(\boldsymbol{v_m}+\epsilon\boldsymbol{x}),\mathcal{U}\big)-D\big(\boldsymbol{PMv_m},\mathcal{U}\big)}{\epsilon} \tag{32}$$

where ϵ is an arbitrary small number and $\boldsymbol{x}$ is a unitary vector in the x direction. $\boldsymbol{P}$ and $\boldsymbol{M}$ are the projection and modelview matrices, as defined in Section 4.5.1. A similar process is applied to each of the motion components.

4.6. Object grasping

In this section we will combine the grasping point computed in Section 4.2 and the arm pose calculated in Section 4.5 in order to move the arm so that it can grasp the object.

After finding the grasping position as a point G^{C_0} in camera space, we move the head to a position where the whole arm is visible, while keeping the object in the field of view. Then, the object position (x, y) is found in this new viewpoint using the method in Section 4.3. We assume that the z coordinate did not change with the gaze shift. Therefore, we use the one calculated previously in G^{C_0}. The grasping point in camera space for the current viewpoint is then $G^{C_1} = \begin{bmatrix} zx & zy & z \end{bmatrix}$.

After this, virtual visual servoing allows us to obtain the transformation $A^A_{C_1}$ that converts points from camera to arm space, so we can obtain the grasping point in arm space as $G^A = A^A_{C_1} G^{C_1}$. To account for small errors in the measurements, we do not move the arm there directly, but take it first to a position which is a few centimeters (15 cm in our experiments) above G^A.

Then, the final step is to bring the arm down so that the object lies between the fingers of the hand. We do this using a simple visual servoing loop. The movement to be performed is purely vertical. Therefore, we only use a single visual feature to control that degree of freedom: the vertical distance between the arm and the grasping point in the image which will be roughly aligned with the vertical axis of the arm. In each iteration, we measure the vertical distance d between the arm and the grasping point in the image, and use this as input for a simple proportional control law:

$$\dot{r}_y = -kd \tag{33}$$

where $\dot{r}_y$ is the vertical velocity of the arm and k is a gain factor.

5. Experiments

5.1. Robustness of Virtual Visual Servoing

In order to evaluate the performance of the virtual visual servoing, we needed a setup where ground truth data was available, so that the error both for the input (edge detection and initial estimation) and the output (estimated pose) is known. Lacking such a setup, we decided to conduct the experiments using synthetic images as input. These images were generated using the same 3D model and rendering system that we use in the visual servoing loop.

The process used in these experiments was as follows:

- Generate a rendered image of the model at a known pose.
- Add some error in translation and rotation to that pose, and use this as the initial pose estimation.
- Run the virtual visual servoing loop with the rendered image as input. In some of the runs, noise was added to the detection of edges, to assess the robustness with respect to certain errors in the edge detection.
- After each iteration, check whether the method has reached a stable point (small correction) and the difference between the detected pose and the known pose is below a certain threshold. If this is the case, consider it a succesful run and store the number of iterations needed.
- If the system does not converge to the known pose after a certain number of iterations, stop the process and count it as a failed run.

We ran three sets of experiments, each of them focusing on the following kind of input error:

- Translational error. We evaluated which range of errors in the translational component of the initial pose estimation allows the system to converge. To that effect, we added, in each run, a translational error of known magnitude and random direction.
- Rotational error. We also evaluated the range of errors in the rotational component of the initial pose estimation for which the system converges to the correct result. For each run, we added a rotational error of known magnitude and random direction.

- Error in edge detection. To simulate the effects of wrongly detected edges in the performance of the system, we added random errors to the edge detection of the input synthetic image. In each run, the detection of a number of pixels was shifted a random amount. An example of the result can be seen in Fig. 10.

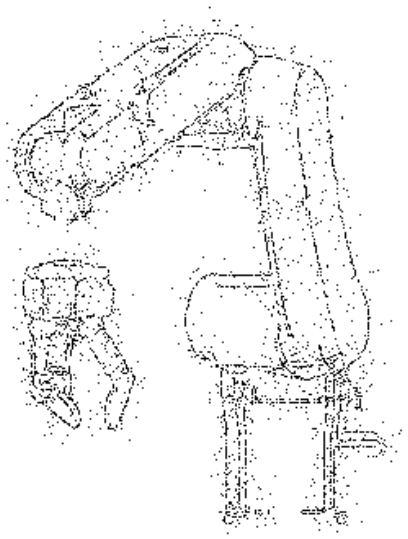


Figure 10: Edge detection with artificially introduced error (in 30% of the pixels).

The performance of the VVS system is measured in (i) the number of runs that failed and (ii) , for the runs that succeeded, the average number of iterations it took to do so. We plot these with respect to the magnitude of the input error. For each value of the input error, we ran 500 visual servoing loops. In each of these runs, the magnitude of the error was kept constant, but the direction (or the particular pixels that were affected in the case of edge detection) was chosen randomly. Also, the whole process was repeated for five different configurations of the joints of the arm.

The rest of the parameters of the process were set as follows:

- The threshold for deciding that the algorithm had converged to the correct pose was 10 mm in translation and 0.5° in rotation.
- The number of iterations after which the run was considered a failure was set to 200.

- For the experiments that evaluate robustness with respect to edge detection, a translation error of 100 mm and a rotation error of 10° was used for the initial pose estimation.

Figure 11 shows the result of increasing the translational component of the error in the estimated initial pose. As we can see, the failure rate is close to 0 for distances below 100 mm, and then starts increasing linearly. This is probably due to 100 mm being in the same order of magnitude as the distance between different edges of the robot which have the same orientation, so the system gets easily lost, trying to follow the wrong edges. We can also observe an increase in the iterations needed for reaching the result.

In Figure 12 we can observe a similar behaviour for the tolerance to errors in the rotational component of the estimated initial pose. Here, the threshold is around 10° and the reason is probably the same as before: this is the minimum rotation that bring edges to the position of other edges.

With respect to the error in edge detection, we can see in Figure 13 that when less than 50% of the pixels are wrongly detected, the system performs almost as well as with no error, and after that the performance quickly degrades. It is also significant that for the runs that converge successfully, the number of iterations is almost independent of the errors in edge detection.

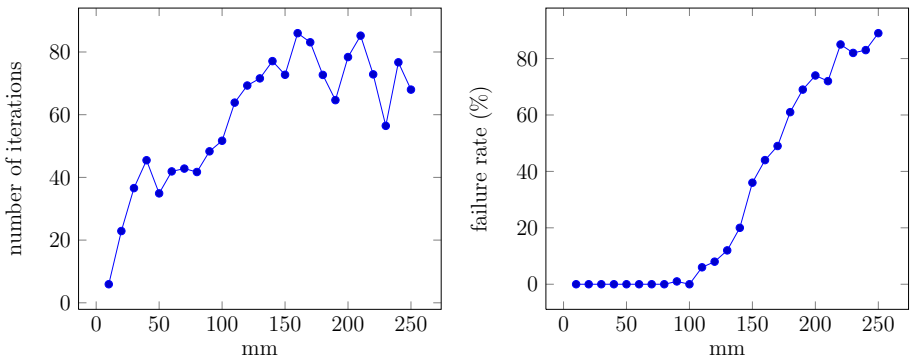


Figure 11: Effects of translational error

5.2. Qualitative Experiments on the Real System

For our experiments with the real robot, we set up a table-top scenario with two randomly placed objects, as can be seen in the last picture in Figure 14. The head was initially looking towards the table, where it could

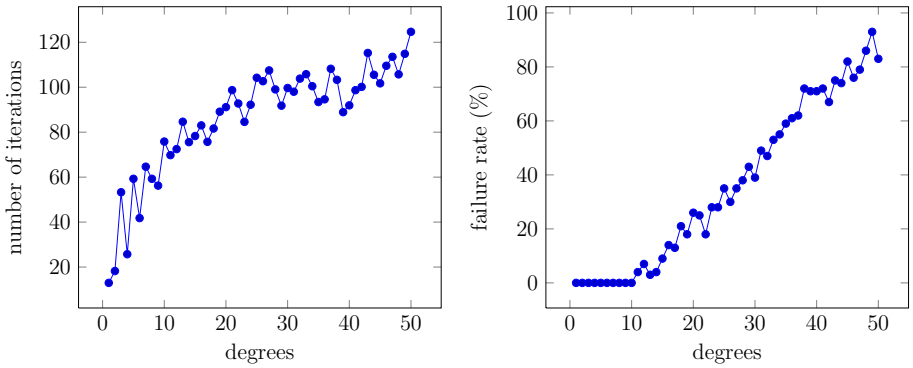


Figure 12: Effects of rotational error

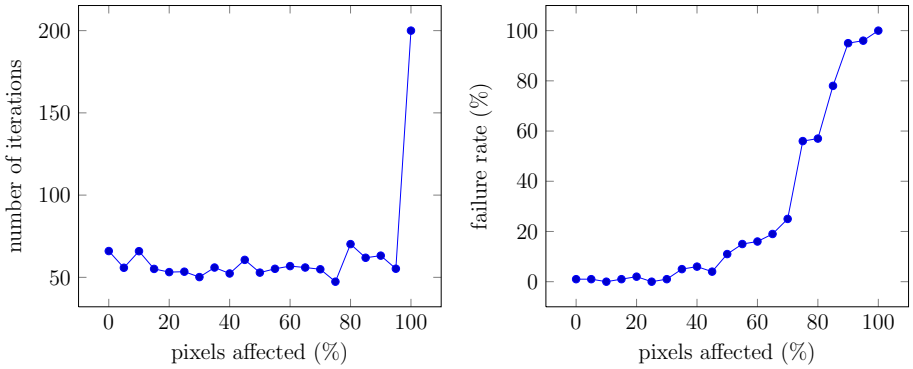


Figure 13: Effects of errors in edge detection

see the objects and find them in the saliency map of the attention system. However, the arm was not fully visible.

We ran the system several times with this setup. In most of these runs, the virtual visual servoing loop did not converge because it could only see a small part of the arm. Therefore, we decided to run virtual visual servoing only after the object is found and after shifting the gaze away from it and towards the arm. With this change, even if the initial estimation for the pose of the arm was significantly different, the pose estimated through visual servoing quickly converged. In Figure 14 we can see, outlined in green, the estimated pose for the arm and hand over the real image.

The segmentation of the object and detection of the grasping point worked well. When running virtual visual servoing with the full arm in view, the hand could be aligned with the object with high accuracy and grasping was

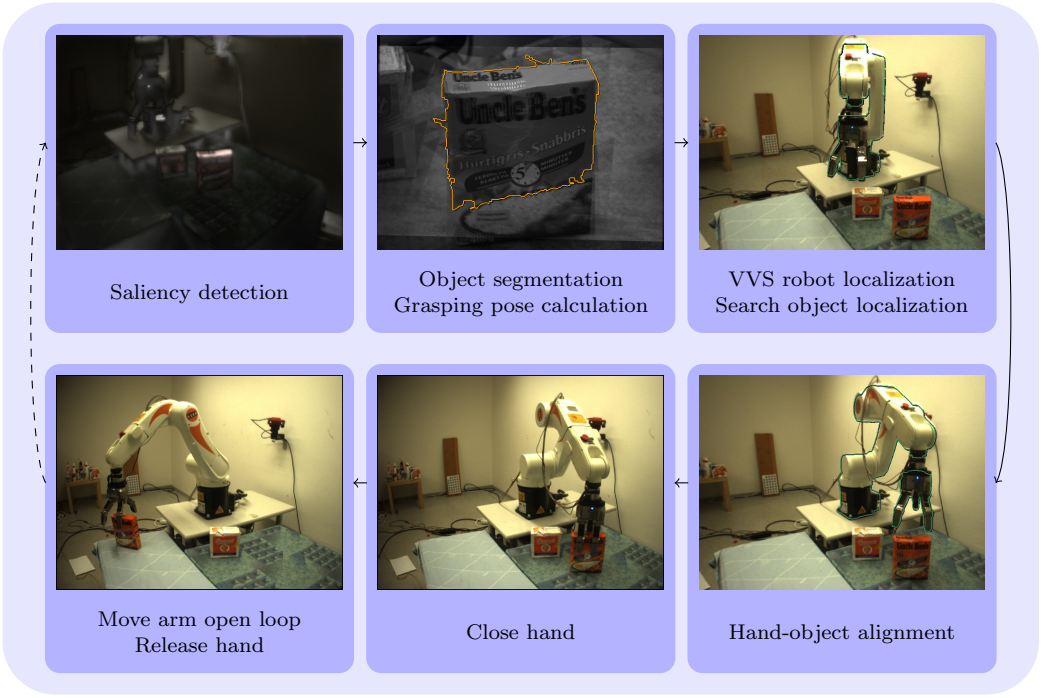


Figure 14: Grasping system diagram. A video with the whole sequence can be found at `http://www.youtube.com/watch?v=e-03Y8_cgPw`

performed successfully in most of the runs.

However, even if they did not affect the performance of the system, there are some problems that arise with the use of virtual visual servoing in real images. For example, as we can see in the upper-right picture of Figure 14, the upper edge of the reconstructed outline does not match exactly with the upper edge of the arm. There are two main reasons for this. First, the illumination of the scene can lead to some edges disappearing, because of the low contrast. This is usually not a problem, because the correct matching of other edges will compensate for this. But in this case, there is a second problem: because of the orientation of the arm with respect to the camera, the outer silhouette has really few edges. As we can see in the picture below that, once the pose of the arm changes, the system can recover and show the correct pose again. As future work, we are considering the possibility of using not only the outer contour, but also try to match some of the internal edges of the model. While this can lead to a less robust system, since outer

edges are more likely to appear as edges in the real image, it can increase performance where the number of edges is low.

6. Conclusions

Grasping and manipulation of objects is a necessary capability of any robot performing realistic tasks in a natural environment. The solutions require a systems approach since the objects need to be detected and modelled prior to an agent acting upon them. Although many solutions for known objects have been proposed in the literature, dealing with unknown objects stands as an open problem.

Our research deals with this problem by integrating the areas of active vision and sensor based control where visual servoing methods represent important building blocks. In this paper, we have presented several ways in which these methods can add robustness and help minimizing the errors inherent to any real system.

First, we showed how visual servoing can be used to automate the offline calibration process. The robot can, without human intervention, generate a pattern to collect data that can then be used to calibrate the cameras and the transformations between the cameras and the arm.

Then, we demonstrated a virtual visual servoing approach to continuously correct the spatial relation between the arm and the cameras. Though in its current state the system works well and is useful in the context of the system, some ideas for future work arise from the experiments we performed. Using the edges as the only feature in the visual servoing loop works well with objects which have a complex outline, such as our arm. However, for other kinds of object, or even some viewpoints of the arm, having a wider range of features might help. These features may include adding textures to the models, or using 3D information obtained via stereo or structured light cameras.

Finally, the last step that brings the arm to the place where the object can actually be grasped also uses visual information to move the arm down to the position where the hand can be closed to grasp the object. There is also room for future improvements here, such as integrating the system with a more complex grasp planner, which could determine the optimal points where the fingers should contact the object. Then, visual servoing could be used to visually bring the fingers to the correct position.

Acknowledgement

This work is supported by the EU through the project GRASP, IST-FP7-IP-215821, Swedish Foundation for Strategic Research and UKF-Unity through Knowledge Fund.

References

[1] J. Glover, D. Rus, N. Roy, Probabilistic Models of Object Geometry for Grasp Planning, in: IEEE International Conference on Robotics and Automation, Pasadena, CA, USA, 2008.

[2] K. Huebner, K. Welke, M. Przybylski, N. Vahrenkamp, T. Asfour, D. Kragic, R. Dillmann, Grasping Known Objects with Humanoid Robots: A Box-based Approach, in: International Conference on Advanced Robotics, 1–6, 2009.

[3] B. Rasolzadeh, M. Björkman, K. Huebner, D. Kragic, An Active Vision System for Detecting, Fixating and Manipulating Objects in Real World, Int. J. of Robotics Research To appear.

[4] P. Azad, T. Asfour, R. Dillmann, Stereo-based 6D Object Localization for Grasping with Humanoid Robot Systems, in: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), 919–924, 2007.

[5] J. Bohg, D. Kragic, Learning Grasping Points with Shape Context, Robotics and Autonomous Systems In Press.

[6] A. Saxena, J. Driemeyer, J. Kearns, A. Y. Ng, Robotic Grasping of Novel Objects, Neural Information Processing Systems 19 (2007) 1209–1216.

[7] J. Speth, A. Morales, P. J. Sanz, Vision-Based Grasp Planning of 3D Objects by Extending 2D Contour Based Algorithms, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, 2240–2245, 2008.

[8] M. Stark, P. Lies, M. Zillich, J. Wyatt, B. Schiele, Functional Object Class Detection Based on Learned Affordance Cues, in: 6th International Conference on Computer Vision Systems, vol. 5008 of *LNAI*, Springer-Verlag, 435–444, 2008.

[9] K. Huebner, D. Kragic, Selection of Robot Pre-Grasps using Box-Based Shape Approximation, in: IEEE/RSJ International Conference on Intelligent Robots and Systems., 1765–1770, 2008.

[10] M. Richtsfeld, M. Vincze, Grasping of Unknown Objects from a Table Top, in: ECCV Workshop on 'Vision in Action: Efficient strategies for cognitive agents in complex environments', Marseille, France, 2008.

[11] D. Aarno, J. Sommerfeld, D. Kragic, N. Pugeault, S. Kalkan, F. Wörgötter, D. Kraft, N. Krüger, Early Reactive Grasping with Second Order 3D Feature Relations, in: ICRA Workshop: From Features to Actions, 319–325, 2007.

[12] N. Bergström, J. Bohg, D. Kragic, Integration of Visual Cues for Robotic Grasping, in: Computer Vision Systems, vol. 5815 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 245–254, 2009.

[13] D. Kragic, H. I. Christensen, Cue Integration for Visual Servoing, IEEE Transactions on Robotics and Automation 17 (1) (2001) 18–27.

[14] T. Asfour, K. Welke, P. Azad, A. Ude, R. Dillmann, The Karlsruhe Humanoid Head, in: IEEE/RAS International Conference on Humanoid Robots (Humanoids), Daejeon, Korea, 2008.

[15] KUKA, KR 5 sixx R850, `www.kuka-robotics.com`, Last Visited Dec'10.

[16] SCHUNK, SDH, `www.schunk.com`, Last visited Dec'10.

[17] X. Gratal, J. Bohg, M. Björkman, D. Kragic, Scene Representation and Object Grasping Using Active Vision, in: IROS'10 Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics, 2010.

[18] A. W. Khan, C. Wuyi, Systematic Geometric Error Modelling for Workspace Volumetric Calibration of a 5-axis Turbine Blade Grinding Machine, Chinese Journal of Aeronautics (2010) 604–615.

[19] J. Felip, A. Morales, Robust sensor-based grasp primitive for a three-finger robot hand, in: Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems, IROS'09, 1811–1816, 2009.

[20] K. Hsiao, S. Chitta, M. Ciocarlie, E. G. Jones, Contact-Reactive Grasping of Objects with Partial Shape Information, in: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), Taipei, Taiwan, 1228–1235, 2010.

[21] A. Ude, D. Omrcen, G. Cheng, Making Object Learning and Recognition an Active Process, Int. J. of Humanoid Robotics 5 (2) (2008) 267–286.

[22] N. Vahrenkamp, S. Wieland, P. Azad, D. Gonzalez, T. Asfour, R. Dillmann, Visual Servoing for Humanoid Grasping and Manipulation Tasks, in: IEEE/RAS Int. Conf. on Humanoid Robots (Humanoids), 406–412, 2008.

[23] A. Ude, E. Oztop, Active 3-D Vision on a Humanoid Head, in: Int. Conf.on Advanced Robotics (ICAR), Munich, Germany, 2009.

[24] K. Welke, M. Przybylski, T. Asfour, R. Dillmann, Kinematic Calibration for Saccadic Eye Movements, Tech. Rep., Institute for Anthropomatics, Universität Karlsruhe, 2008.

[25] V. Lepetit, J. Pilet, P. Fua, Point matching as a classification problem for fast and robust object pose estimation, in: CVPR, II: 244–250, 2004.

[26] T. Drummond, R. Cipolla, Real-Time Visual Tracking of Complex Structures, IEEE Trans. PAMI 24 (7) (2002) 932–946.

[27] V. Kyrki, D. Kragic, Integration of model-based and model-free cues for visual object tracking in 3D, in: IEEE International Conference on Robotics and Automation, ICRA'05, 1566–1572, 2005.

[28] D. Kragic, V. Kyrki, Initialization and System Modeling in 3-D Pose Tracking, in: In IEEE International Conference on Pattern Recognition 2006. ICPR'06, Hong Kong, 643–646, 2006.

[29] A. I. Comport, D. Kragic, E. Marchand, F. Chaumette, Robust Real-Time Visual Tracking: Comparison, Theoretical Analysis and Performance Evaluation, 2841 – 2846, 2005.

[30] J. Romero, H. Kjellström, D. Kragic, Hands in action: real-time 3D reconstruction of hands in interaction with objects, in: ICRA, IEEE, 458–463, 2010.

[31] B. León, S. Ulbrich, R. Diankov, G. Puche, M. Przybylski, A. Morales, T. Asfour, S. Moisio, J. Bohg, J. Kuffner, R. Dillmann, OpenGRASP: A Toolkit for Robot Grasping Simulation, in: SIMPAR '10: Proceedings of the 2nd International Conference on Simulation, Modeling, and Programming for Autonomous Robots, 2010.

[32] M. Björkman, D. Kragic, Active 3D Scene Segmentation and Detection of Unknown Objects, in: IEEE International Conference on Robotics and Automation, 2010.

[33] M. Björkman, D. Kragic, Active 3D Segmentation through Fixation of Previously Unseen Objects, in: British Machine Vision Conference, to appear, 2010.

[34] Z. Zhang, A Flexible New Technique for Camera Calibration, IEEE Transactions on Pattern Analysis and Machine Intelligence 22 (2000) 1330–1334, ISSN 0162-8828.

[35] M. A. Butt, P. Maragos, Optimum design of chamfer distance transforms, IEEE Transactions on Image Processing 7 (10) (1998) 1477–1484.

[36] D. M. Gavrila, Multi-Feature Hierarchical Template Matching Using Distance Transforms, in: ICPR, Vol I: 439–444, 1998.

[37] M. Liu, O. Tuzel, A. Veeraraghavan, R. Chellappa, Fast directional chamfer matching, in: CVPR, IEEE, 1696–1703, 2010.

[38] A. I. Comport, É. Marchand, M. Pressigout, F. Chaumette, Real-Time Markerless Tracking for Augmented Reality: The Virtual Visual Servoing Framework, IEEE Trans. Vis. Comput. Graph 12 (4) (2006) 615–628.

[39] S. Klose, J. Wang, M. Achtelik, G. Panin, F. Holzapfel, A. Knoll, Markerless, Vision-Assisted Flight Control of a Quadrocopter, in: IROS, 2010.

[40] T. Drummond, R. Cipolla, Real-Time Visual Tracking of Complex Structures, IEEE Trans. Pattern Anal. Mach. Intell 24 (7) (2002) 932–946.

[41] G. Borgefors, Distance Transformations in Digital Images, Computer Vision, Graphics and Image Processing 34 (1986) 344–371.

[42] S. Hutchinson, G. Hager, P. Corke, A tutorial on visual servo control, IEEE Transactions on Robotics and Automation 12 (5) (1996) 651–670.