SEVENTH FRAMEWORK PROGRAMME

Project Acronym:	Xperience
Project Type:	IP
Project Title:	Robots Bootstrapped through Learning from Experience
Contract Number:	270273
Starting Date:	01-01-2011
Ending Date:	31-12-2015

XPERIENCE.ORG

Deliverable Number:	D2.1.1
Deliverable Title:	Sensorimotor Experience (I): Report or scientific publication on low-level perceptual representations of the system's experience with focus on their use in reaching and grasping.
Type (Internal, Restricted, Public):	PU
Authors:	F. Wörgötter, E. Aksoy, J. Papon, M. Tamosiunaite, J.Piater, D. Kraft, N. Krüger, G. Metta, T. Asfour, K. Welke, D. Gonzalez, R. Dillmann
Contributing Partners:	UGOE, SDU, UIBK, KIT, IIT

Contractual Date of Delivery to the EC:	31-01-2012
Actual Date of Delivery to the EC:	31-01-2012

Contents

Chapter 1

Executive Summary

This deliverable summarizes the efforts of the consortium on developing low-level perceptual representations. These include visual, visuo-haptic and motor representations.

In Chapter 2 we present results on visual data representations. We consider several approaches to structure visual information. One approach is to segment the visual scene and consider relations between segments. This approach is presented in Section 2.1. A GPU-based system allows achieving real-time segmentation. Later graphs of geometrically neighbouring segments are formed and graph tracking is performed over time. The second approach is based on the extraction of contours and surfaces from the image and analysis of relations of those entities. This approach is described in section 2.2. The emphasis is on representations which are color and pose invariant but show geometric similarities of shapes. The third is model-based approach. This approach is described in section 2.3. Here part-based object model building and object detection in the visual scene as well as learning parametric abstractions of shapes using kernel methods are presented.

The outcomes of the work on visual representation are the following:

- Real-time image analysis and segmentation system is developed;
- Segment graph extraction and graph tracking over time is implemented allowing to segment sequences of observations into meaningful pieces;
- Methods for analyzing relations between contours and surfaces in a scene are developed;
- Features for describing geometric shapes are proposed;
- Part-based object modeling method is developed and is aimed at part-based generalization;
- Methods for model-based object detection in a scene and pose estimation are developed;
- Method for learning shapes using kernel methods is developed.

Chapter 3 considers the development of visuo-haptic object representations. As in some situations shape and texture estimation as well as object-background separation is very complicated, this problem is being solved by establishing common reference frame, which allows binding visual and haptic information and thus enriching visually extracted information by tactile sensory data. The inclusion of haptic information allows performing a reliable figure-ground segmentation even in such situations, where object and background are visually ambiguous.

The outcomes of work on visio-haptic representations are the following:

- Strategy for haptic exploration and blind grasping of unknown objects is developed;
- Algorithm for binding visual and haptic information is developed;
- Algorithm for establishing contact point with the object is developed;
- Tactile exploration strategy based on contour following is proposed;

- All the above is implemented on the humanoid robot ARMAR-III with preliminary evaluation of performance.

Chapter 4 considers motor representations with the emphasis on reaching and grasping. In section 4.1 the iCub architecture for reaching including low-level force control with dynamics compensation and optimization-based kinematics is presented. The implementation allows controlling the arm as well as other body part (e.g. waist). Another way to represent motor actions/experiences (especially reaching) is using Dynamic Movement Primitives (DMPs). However, the work on defining reaching movements using DMPs within the first project year went beyond simple reaching representations, towards agent-cooperative reaching regimes, and is reported as a whole in D4.1.1. For grasping the grasps using three-finger hand were newly defined (in addition to earlier existing ones for the parallel gripper). Grasps were based on contour as well as surface information. Large number of grasps was executed in a dynamic simulator this way collecting statistics about visuo-motor grasping experience.

The outcomes of work on motor representations are the following:

- Method of computation of robot dynamics and kinematics based on graph representation is developed;
- Application of the method for definition of the reaching action is presented.
- Method is implemented and evaluated on iCub robot;
- DMP based representations have gone beyond the extent of this deliverable by including agent cooperation components;
- Grasp definitions based on contour and surface information are developed;
- Large grasp experience database is collected using dynamic simulator.

Chapter 2

Creating rich representation of visual data

2.1 Scene graphs and tracking

2.1.1 Extracting scene graphs using Oculus system

This section presents real-time open-source vision system for scene graph extraction [PAAW12]. Graphs are extracted from live or recorded video. Oculus is a plugin shell which provides an API for interacting with a graphical user interface (GUI), memory management system, and visualization components. It enables development and use of complex vision pipelines integrating any number of algorithms. Individual algorithms are implemented using modular plugins, allowing integration of methods developed independently and rapid testing of new vision pipeline configurations. The architecture exploits the parallelization of graphics processing units (GPUs) and multi-core systems to speed processing and achieve real-time performance. Additionally, the use of a global memory management system for frame buffering permits complex algorithmic flow (e.g.feedback loops) in online processing setups, while maintaining the benefits of threaded asynchronous operation of separate algorithms.

Graphs are extracted in real-time using a system configuration consisting of four plugins. The overall execution flow of the system, as well as examples of the outputs of the plugins is shown in Figure 2.1. The first plugin in the pipeline acquires visual and depth data using a Kinect camera and the OpenNI library [3]. Video can also be acquired using a stereo camera rig, but depth data derived from stereo images is generally less accurate than data from an active sensor, such as the Kinect.

Next, optical flow is computed on the GPU using the method of Pauwels[27]. This is a phase-based algorithm [16], which tracks the temporal evolution of equi-phase contours by taking advantage of phase constancy. Differentiation of the equi-phase contours with respect to time yields spatial and temporal phase gradients. Integrating the temporal phase across orientation yields optical flow fields. The plugin uses the five most recent frames to compute optical flow in the case of online video, but can also use "future" frames when working with recorded movies (to improve the quality of the results). When using a stereo camera setup, sparse disparity maps are computed in a separate plugin using a technique similar to optical flow [27]. Rather than use temporal phase gradients, the disparity algorithm relies on phase differences between stereo-pair rectified images.

The depth, visual, and optical flow data are then used to perform combined segmentation and tracking on the GPU using the method of Abramov et al.[5]. This accomplishes two goals; first, it partitions the image into labeled regions, as seen in the right-most column of Figure 2.1, and second, it determines correspondences between successive frames to maintain consistent labeling. The segmentation algorithm applies the Potts model in such a way that superparamagnetic phase regions of aligned spins correspond to a natural partition of the image data. Initial spins are assigned to pixels randomly, and then a Metropolis-Hastings algorithm with annealing [5] is used to iteratively update the spins until an equilibrium state is reached. In addition to segmentation, the plugin maintains consistent labels for objects from frame to frame. This is accomplished by transferring spins between frames using optical-flow[5]. As such, only the first frame is actually initialized at random; subsequent frames are initialized using a forward-propagated version of the previous frame's equilibrium spins.

Finally, graphs are constructed by analzying the spatial relations of segments, as in the work of Aksoy et al.[7, 6]. This process is described in detail in the next section.

2.1.2 Graph tracking

Following the extraction of segments (see previous section), we analyze the spatial relations between each segment pair. We denote spatial relations by $\rho_{i,j}$ in which i and j are the segment numbers. Note that spatial relations are symmetric, i.e. $\rho_{i,j} = \rho_{j,i}$. We define two basic relations between segments: *Touching* and *Non-touching* each of which refers to image segments that are physically neighbors or not in 3D domain, respectively. Such spatial relations are directly referring to primitive manipulations.

Once the image sequence has been segmented and spatial relations have been extracted, we represent the scene by undirected and unweighted labeled graphs. The graph nodes are the segment labels and plotted at the center of each segment. Nodes are then connected by an edge if segment relations are *Touching*.

Scene graphs, such as those depicted in Fig. 2.2, represent spatial relations between nodes in temporal domain. Unless touching and non-touching events happen, the scene graphs remain topologically the same. The only changes in the graph structures are the node positions or the edge lengths depending on the object trajectory and speed. Any touching or non-touching of segments corresponds to a change in the main structure of the scene graphs. Therefore, those changes in the graphs can be employed to define manipulation primitives. We apply an exact graph-matching method in order to extract the main graphs by computing the eigenvalues and eigenvectors of the adjacency matrices of the graphs [33]. A change in the eigenvalues or eigenvectors then corresponds to a structural change of the graph. Details of this approach has been explained in detail elsewhere [7, 6].

Fig. 2.2 depicts some of main graphs calculated from a scenario "Moving Boxes" in which a hand is picking a box and placing on top of another. While the hand is performing the picking and placing manipulation, graphs are changing topologically, e.g. some nodes and/or edges appear and/or disappear.

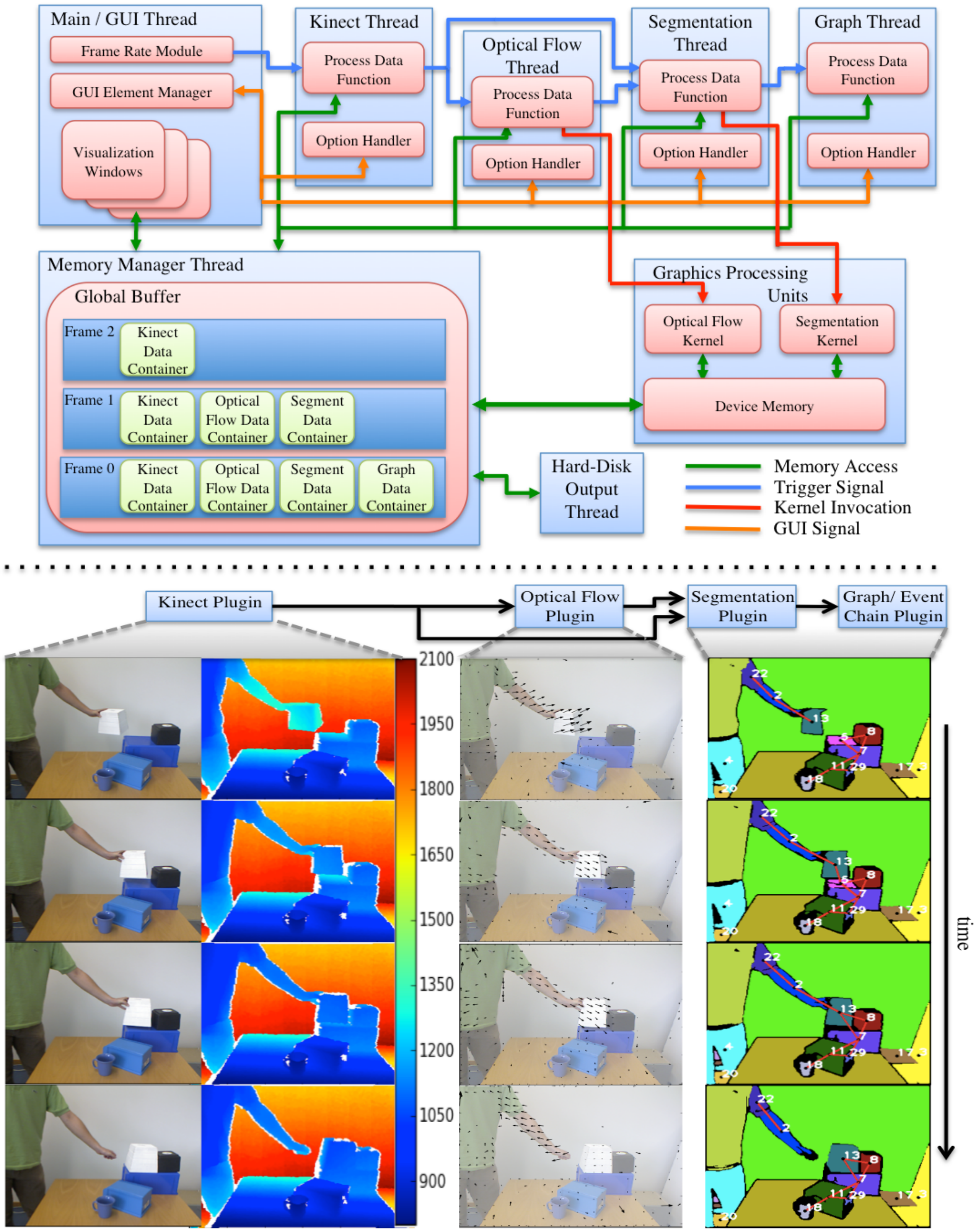


Figure 2.1: Overview of the system architecture and demonstration system output for four frames. The colums show output from the different components; from left to right, Kinect image, Kinect depth (in mm), optical flow, and graphs overlaid on segmentation output.

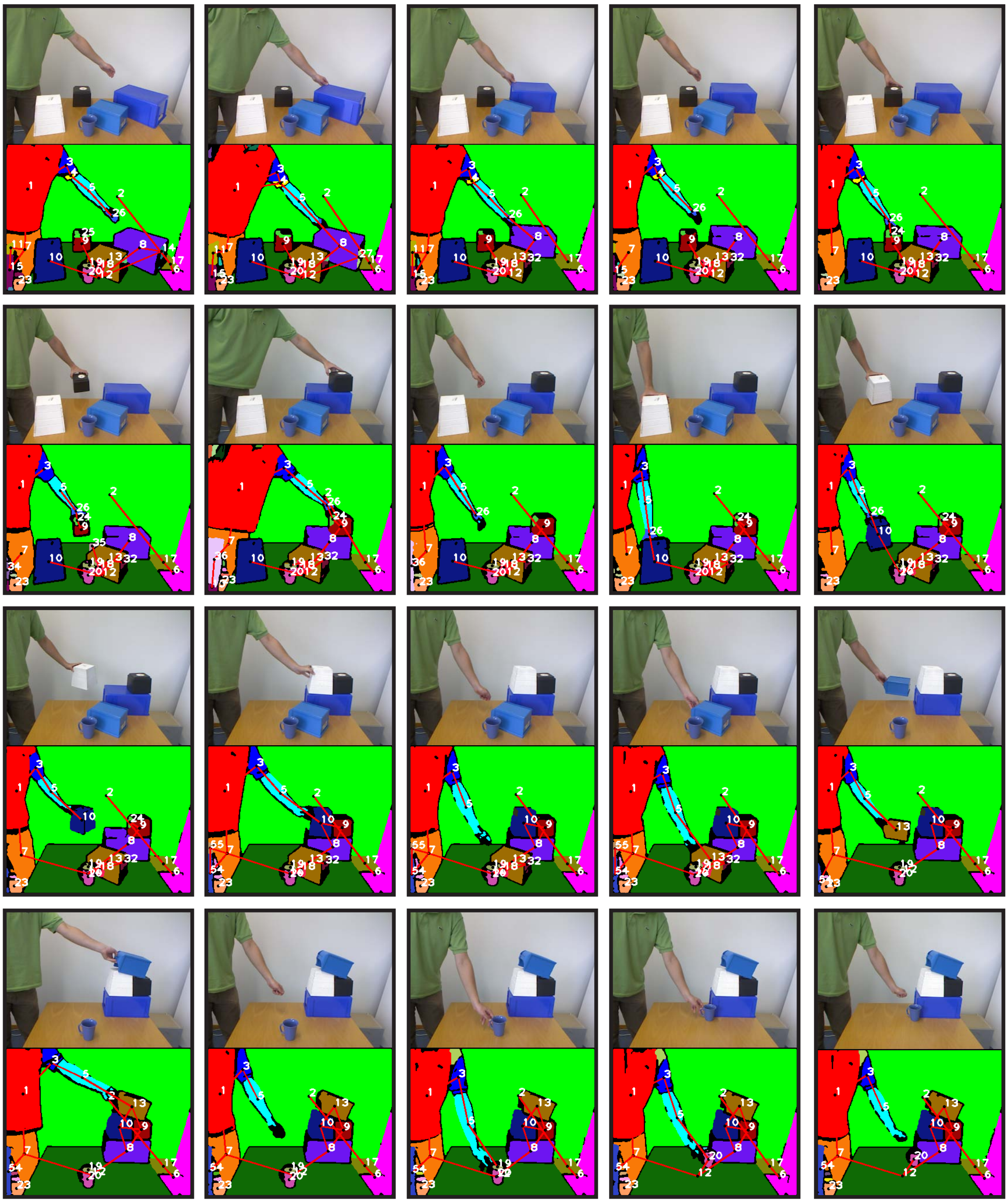


Figure 2.2: *Sample original frames with respective segments and scene graphs from a scenario Moving Boxes*

2.2 Extracting view-point invariant representations based on geometric and appearance relations

This section present a view-point invariant representations based on geometric and appearance relations. The basis for this representation is formed by the Early Cognitive Vision (ECV) System [29] and its recent extension to texture features [24]. The ECV system allows for the extraction and classification of image structures (line-segment, junction and texture) from images and later on their combination within stereo and grouping processes. In this way the system forms a hierarchical image representation. Figure 2.3 gives an example for the line-segment (contour) and the texture (surface) domain.

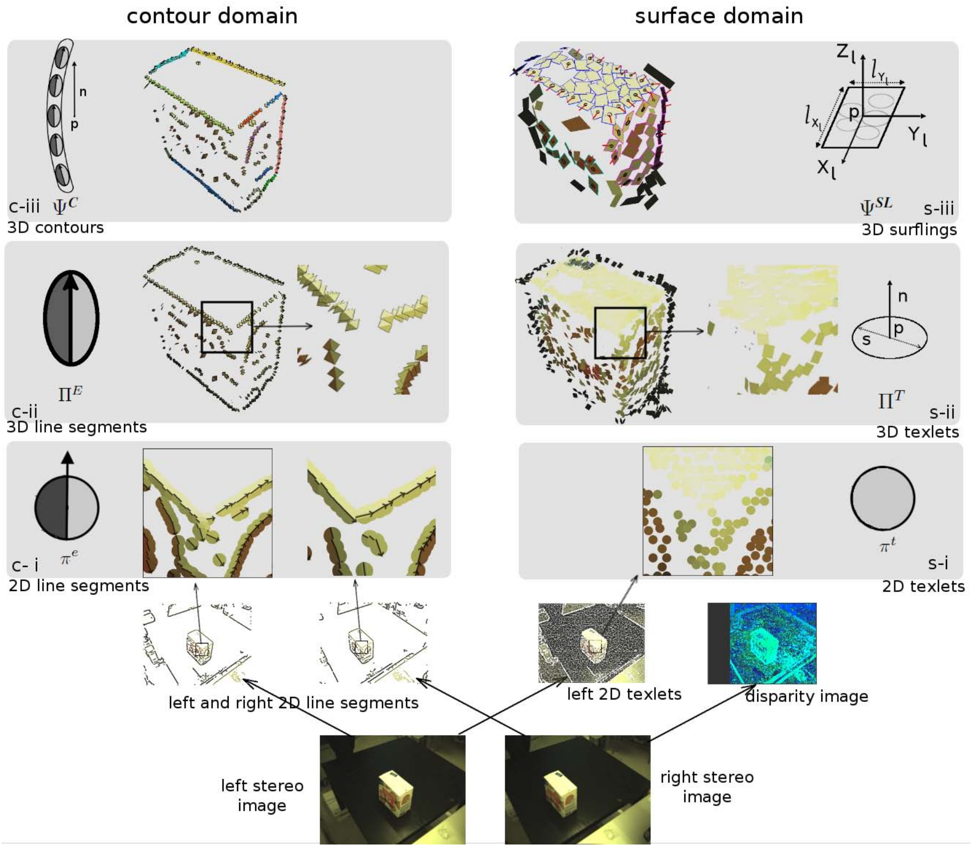


Figure 2.3: The hierarchical representation of edge and texture information in the ECV system. Based on an example stereo image pair 2D line segments for the left and the right image, 2D texlets for the left image and a disparity image are extracted. (c-i) 2D line segment details. (c-ii) 3D line segments. (c-iii) 3D contours. (s-i) 2D texlet details. (s-ii) 3D texlets. (s-iii) 3D surflings.

In two hierarchies' complementary visual information is represented on different levels of granularity together with the associated uncertainties and confidences. On all levels geometric and appearance information is coded explicitly allowing to access this information separately and to link between the different levels. The geometric and appearance information of the individual features allows us to define relations between these features. Some important feature relations here are for example Euclidean distance, normal distance, angle (binary relations) and the individual color values (a unary relation). A complete overview of the different relations is given in [24].

This visual representation has a number of interesting properties in the context of classical vision problems such as object recognition, pose estimation as well as a number of learning problems such as grasp affordance learning. The view-point invariant representation based on geometric and appearance relations makes use of three of those properties, namely: (1) the separation between geometric and appearance information, (2) view point invariance, and (3) the richness in terms of providing a large set of relevant attributes.

The viewpoint-invariance of our representation can be shown by demonstrating the system's ability to maintain stable appearance and geometric attributes and relations under viewpoint transformation. In the below work we studied this stability by means of a histogram approach. The work has been performed in a simulated setup to have optimal control over object properties. To get a rather complete representation three simulated cameras placed around the object were used.

The objects in study here are three variations of a 100x150x200 mm box. The two first objects are closed boxes with different colors; white and green respectively. They are positioned with a relative 90° rotation around the high axis. The third object is an open box (i.e., the same as the first box except a missing top surface) with gray texture and with the same pose as the first object. Figure 2.4 shows histograms corresponding to a subset of the surfling attributes and some second order relations extracted from the stereo images for the three objects.

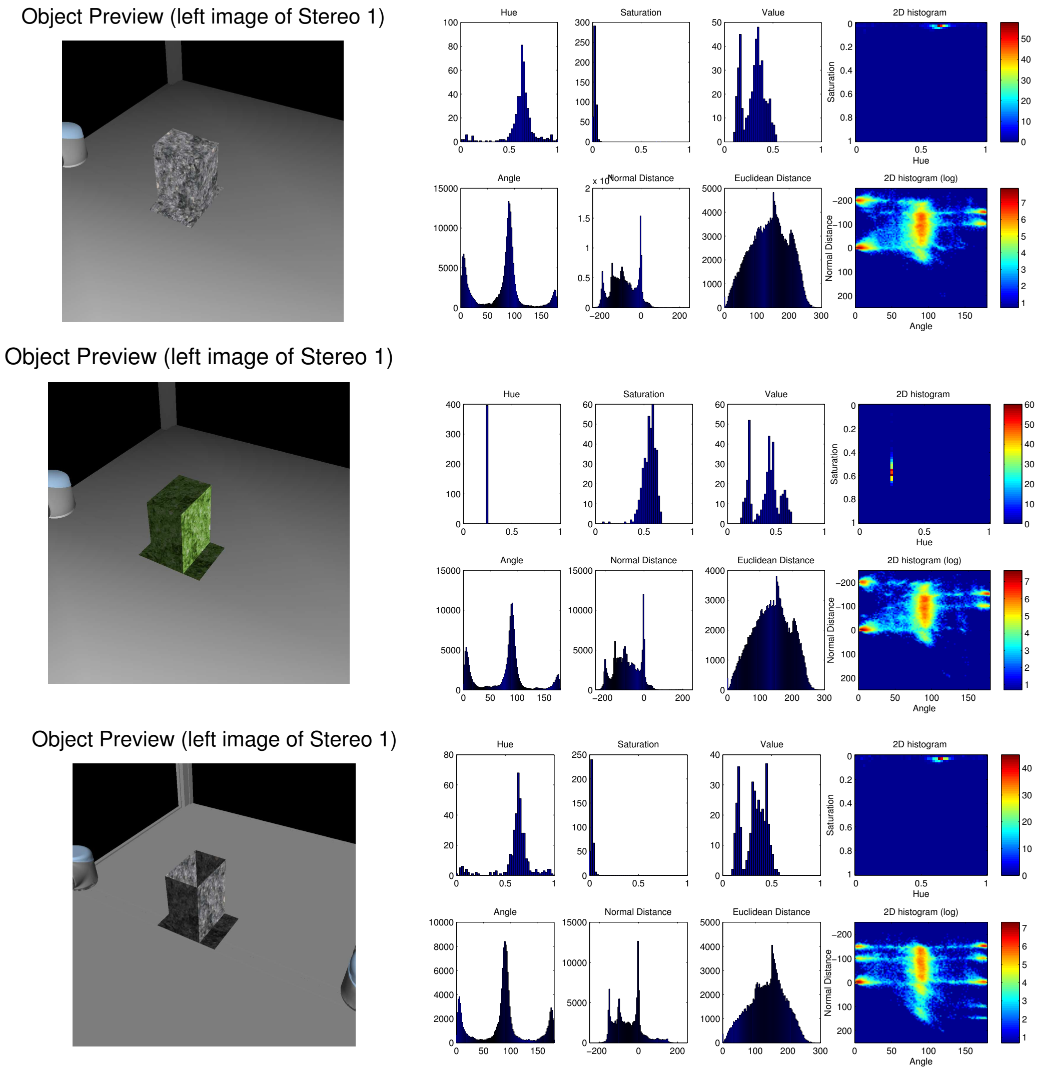


Figure 2.4: Three different scene configurations and corresponding histograms. The histogram blocks for each scene present the following components: **(top row)** Three appearance histograms representing the hue (H), the saturation (S) and the value (V) color information of the extracted surflings, while the right most histogram shows the second order hue vs. saturation histogram. **(bottom row, left)** shows three aspects of the geometric information: (1) The histogram corresponding to the angles of pairs of surflings, (2) their normal distance and (3) their Euclidean distance relation are shown. **(bottom row, right)** The histogram spanned by the angle between pairs of surflings as well as their normal distance is shown.

It can be easily noticed that the appearance histograms of the first and the third object are similar. Analogously, the histograms representing the geometric relations of the first and and the second objects are very similar. This similarity is due to the fact that the first and the third object have the same appearance while the first and the second have the same geometry. The difference in color of the second object with respect to the first and the third object is reflected in the appearance histograms being different.

Comparing the first and second object, it is noticeable that very similar relation histograms are obtained despite the difference in pose. The geometric relations, being view point invariant by definition, also show this property when being extracted from stereo images where the noise introduced by the uncertainties in this controlled environment is rather limited.

In both cases, the two-dimensional angle / normal distance histograms reveal three peaks at around $(0°, -200mm), (180°, -100mm)$, and at $(180°, -150mm)$. These peaks correspond to the dimensions of the box and the distances of the parallel planes (negative values indicate an outward direction). Peaks at a certain normal distance with angle of 0 degree represent parallel surfaces pointing in the same direction (i.e., the table and the top surface) and of 180 degrees when they are pointing in opposite directions (the four side planes).

When comparing the third object with the two others, it can be seen that the peak at $(0°, -200mm)$ does not appear anymore while five other peaks are introduced at $(0°, -100mm)$, at $(0°, -150mm)$, at $(180°, 0mm)$, at $(180°, 100mm)$, and at $(180°, 150mm)$. The object in this case is a box that has no top surface, and this explains the disappearance of the peak at $(0°, -200mm)$. Furthermore as a result of having an open box in the third box, the inside faces of the surfaces can also be seen and their surflings descriptors can be extracted. Therefore, surflings pointing in the same direction (the outside face of a surface and the inside face of the parallel surface) can be obtained and this explains the two new peaks at 0°. In addition to that. the two peaks at $(180°, 100mm)$ and at $(180°, 150mm)$ represent the case where surflings are pointing in opposite directions (the inside faces of the parallel surfaces). In addition there is a large peak at $(180°, 0mm)$ representing surfling pairs on the same surface at opposite sides.

The above scenario shows that very similar histograms for color and geometric are obtained when the objects have the same color despite their geometrical and pose differences. It also shows that the geometric relations code the properties of the object in a stable way despite the differences in the pose or the color. Hence, the ECV system can provide a view-point invariant appearance and geometric representation of objects.

This example also shows that the histograms show a clear difference in case of open and closed objects. This allows us to distinguish a closed box from an open box and can be used to derive a predicate for open/closed.

2.3 Visual modeling of objects

2.3.1 Pose estimation in 2D images using probabilistic 3D object models

This section presents the work realized on pose estimation in 2D images. More details were presented in [TP11]. We focused on the use of a single, monocular image as the source of scene observations, and known, 3D models of the object considered. The state-of-the-art methods in this domain generally rely on matching characteristic, local visual features between the observations of the scene and the model of the object. This approach, although efficient with textured objects or otherwise matchable features, would fail when considering non-textured objects, or visual features that cannot be as precisely located as the texture patches or geometric features used in the classical methods. Our method therefore makes few assumptions about the type of features used, and it does not rely on establishing specific matches between features of the model and of the observed scene. For this purpose, we represent both the object model and the 2D observations of a scene as probabilistic distributions of visual features. The model is built from 3D observations that can be provided by any external, independent system. One of the main interests of the proposed method, in addition to the genericity of the underlying principles, is its ability to effectively handle non-textured objects. The general method itself does not make particular assumptions about the type of features used, except that they must have a given, although not necessarily exact, position in space, and they must be potentially observable in a 2D view of the object.

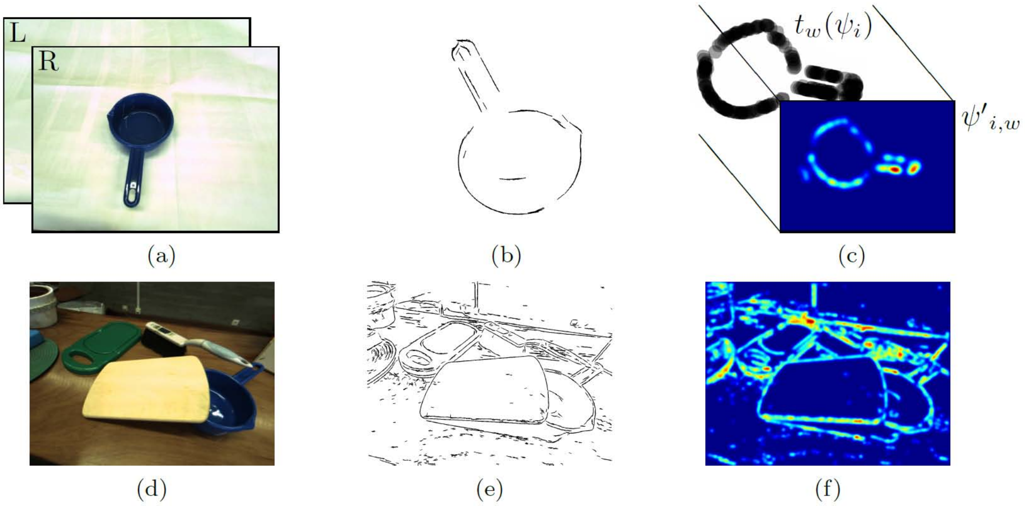


Figure 2.5: Method for pose estimation in 2D images using edge segments. *Top row, object model:* (a) stereo images of object i used to build the model; (b) 3D edge segments that compose the model ψ_i; (c) probabilistic model ψ_i transformed to pose w, and its projection $\psi'_{i,w}$ in 2D (blue and red represent resp. lowest and highest probability densities). *Bottom row, scene:* (d) image of a scene; (e) 2D edge segments used as observations; (f) probabilistic representation of observations, to which projected models such as $\psi'_{i,w}$ are matched under all poses w using MCMC.

Object model

The 3D observations used to build the model are provided by an external system that performs stereopsis on a single pair of images. Such a model can thus be quickly and automatically learned, at the expense of imprecision and imperfections in the model. This again motivates the use of a probabilistic distribution of features as the object model. An object is represented as a conjunction of parts and their poses relative to the object. Each part is represented as a spatial distribution of its constituent features, again in object-relative pose space. Thus, object detection and pose estimation amounts to globally-consistent matching of model features to scene observations via probabilistic inference.

Scene observations

In order to demonstrate the capabilities of the proposed method at handling textureless objects, we apply it using local edge segments as observations. Practically, such features cannot be precisely and reliably observed in 2D images, e.g., due to the ambiguity arising from multiple close edges, 3D geometry such as rounded edges, or depth discontinuities that change with the point of view. Such problems motivate the probabilistic approach used to represent the scene observations.

Pose estimation

The object and observation models presented above allow us to estimate the pose of a known object in a cluttered scene. This process builds on the idea that the 2D projection of the 3D probability distribution defining the object model can be used as a “template” over the observations, so that one can easily measure the likelihood of a given pose. A likelihood function for the pose can thus be rigorously defined, using the probability distributions of the model and of the scene. The maximum-likelihood pose is sought using an iterative method based on the Metropolis-Hastings MCMC algorithm (Figure 2.5).

Additionally, the proposed method provides a rigorous framework for integrating evidence from multiple views, yielding increased accuracy with only a linear increase of computation time with respect to the number of views. As opposed to the classical way stereo images are used, our approach does not seek matches between the two images, as stereopsis does, and it can thus handle arbitrarily wide baselines.

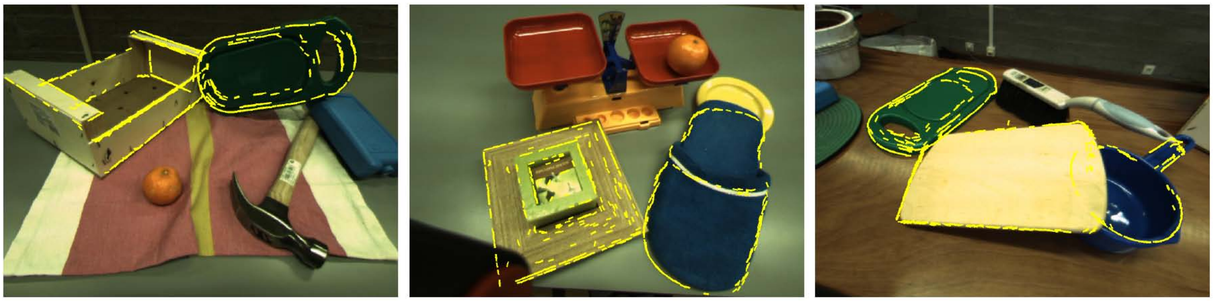


Figure 2.6: Results of pose estimation (using a single view), with model features reprojected onto the input image.

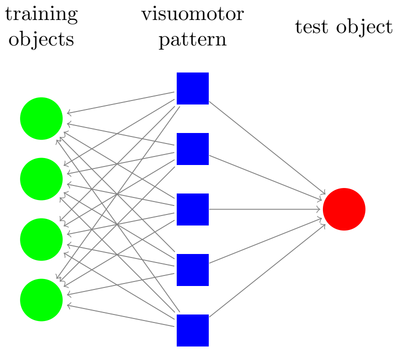


Figure 2.7: Parts can be considered as visuomotor patterns of objects.

Performance

We validated the proposed approach on two publicly-available datasets. One dataset allowed quantitative evaluation; the result of an experiment was compared to the results of an existing method, and showed an advantage in performance for our method. The pose estimation process was also evaluated with success on scenes with clutter and occlusion (Figure 2.6).

For more detail please refer to the associated publication [TP11].

2.3.2 Building part-based object models using Kinect data

Part-based models have been widely used in modeling 2D objects. Their advantages over other models can be mainly considered from two perspectives: (1) They provide a structure with good interpretation consistent with evidence that the human visual mechanism is also part-based; (2) part-based structure can be more robust to the circumstances in which objects are partially occluded. In addition, in the case of 3D objects modeling, besides the two issues mentioned above, another important strength of part-based models is that the grasping (or other manipulation) knowledge can also be generalized based on composing parts. The underlying idea is that objects that share similarities in corresponding parts should also hold similar grasping properties. One intuitive example is that a handle is a common part that appears in knifes, forks and spoons, and all these objects can be grasped in quite similar manner. In this way, we can avoid specific grasping training for different individual objects. Instead, the grasping properties of a novel object can be inferred by detecting parts shared with previously-trained objects, that is, by part-based grasping generalization. More abstractly, part-based grasping generalization can be considered as the extraction of general *visuomotor patterns* relating the *appearance* and *grasping* properties of parts via explicit representations (see Figure 2.7).

In our 3D object modeling case, the Kinect is used to obtain depth data (thus the color information is lost), and for simplicity of interpretation and visualization, they are converted to 3D point clouds. *Kernel density estimation* (KDE) can be applied on 3D point cloud representations, which provides a probabilistic form of objects, and thus the object pose estimation and detection can be implemented within a probabilistic framework. More concretely, for each point x, the surface normal $\theta \in S^2$ of

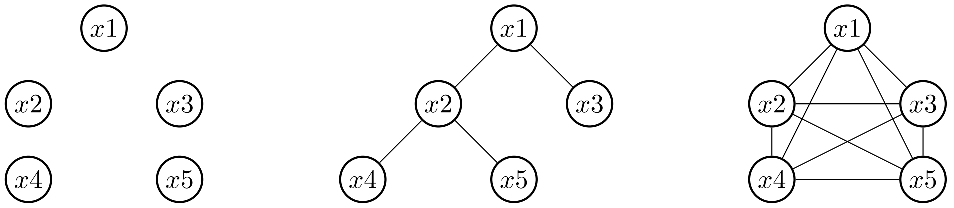


Figure 2.8: Left: *bag of parts*, in which all parts are considered independently; Center: *tree structure*, in which parts are organized in a hierarchy; Right: *fully connected structure*, in which all parts are connected to each other.

its neighboring region can be computed, and combined with position $\lambda \in \mathbb{R}^3$. A point cloud can be represented as $\{x^{(i)}\}_{i=1}^N = \{\lambda^{(i)}, \theta^{(i)}\}_{i=1}^N$. By applying KDE on points $x \in \mathbb{R}^3 \times S^2$, a Gaussian kernel is established based on the position, and a von Mises-Fisher distribution kernel is constructed on the normal orientation. Thus the probabilistic form of a point cloud can be written as:

$$\psi(x) = \sum_{i=1}^{N} \mathbf{K}(x^{(i)}) \tag{2.1}$$

$$\mathbf{K}(x^{(i)}) = \mathbf{N}(\lambda^{(i)}, \sigma_\lambda) \times \Phi(\theta^{(i)}, \sigma_\theta) \tag{2.2}$$

where $\mathbf{N}(\lambda^{(i)}, \sigma_\lambda)$ is the position-based Gaussian kernel, and $\Phi(\theta^{(i)}, \sigma_\theta$ is the orientation-based von Mises-Fisher kernel, and σ_λ and σ_θ denote the bandwidth of position and orientation kernels.

Generally, the part-based models can be categorized according to connectivity structure between parts. Three popular models are: *bag of parts*, *tree structure*, *fully-connected structure* (see Figure 2.8). Since the complexity of fully connected models is exponential in the number of parts, they are difficult to use in practice. Therefore, only bag-of-parts and tree models are investigated in our experiments:

- *Bag of parts model:* according to (2.1), each part is mathematically represented as a distribution $\varphi_i(x)$ using KDE on point cloud. Scene distributions $\psi(x)$ can be constructed in the same way. The pose estimation and simultaneous detection (these two tasks can be done in one shot) can be done by finding the optimal pose $w*$ which maximizes the cross correlation between the scene and model distributions:

  $$w^* = \operatorname*{argmax}_{w} \int \psi(x)[t_w(\varphi_i(x))]\,\mathrm{d}x \tag{2.3}$$

  where $t_w(\varphi(x))$ is the distribution of the part after the corresponding transformation of pose w.

  However, in this way, the recovered poses of the parts will probably differ from each other. Although parts are modeled independently, they should be constrained to arrive at a consistent pose. One straightforward way to compute a unique global optimal pose for all parts (and thus also for the object) is as follows:

  $$w^* = \operatorname*{argmax}_{w} \prod_{i}^{\#(parts)} \int \psi(x)[t_w(\varphi_i(x))]\,\mathrm{d}x \tag{2.4}$$

  Since there is no closed form to compute (2.4), *Markov Chain Monte Carlo* (MCMC) and *simulated annealing* are applied to obtain an estimation of global optimal solution. One test example can be seen in Figure 2.9.

- *Tree-structured model:* Different from the bag-of-parts model, tree-structured models organize all parts in a hierarchical structure. The model can be constructed top-down (segmentation) or bottom-up (merging); see Figure 2.10. Here, bottom-up construction is used: lower-level parts are merged into to higher level parts, and the merging procedure repeats until the whole object is composed. In this way, pose estimation can be done at different part-complexity levels, and a *hierarchical conditional random field* of all estimated poses can be constructed on the 3D point cloud observations:

  $$p(w_0, w_1, \ldots, w_N|O) = \frac{1}{Z} \prod_{i,j \in \text{connection}} \pi(w_i, w_j) \prod_{k} \tau(w_k, O) \tag{2.5}$$

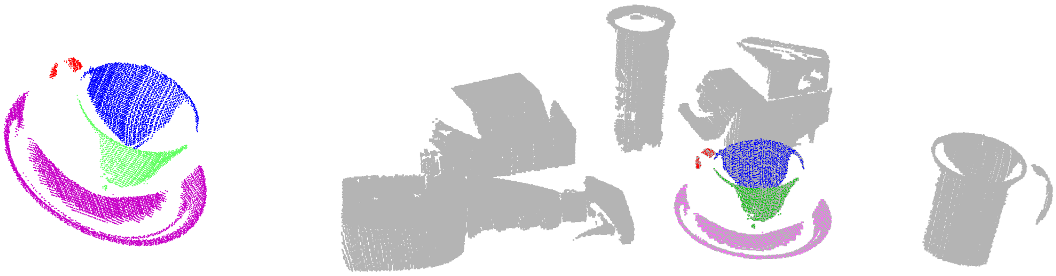


Figure 2.9: Left: a cup is decomposed into 4 parts, and each part is modeled as a 3D point cloud; Right: pose estimation and detection of the cup within clutter by using a bag-of-parts model.

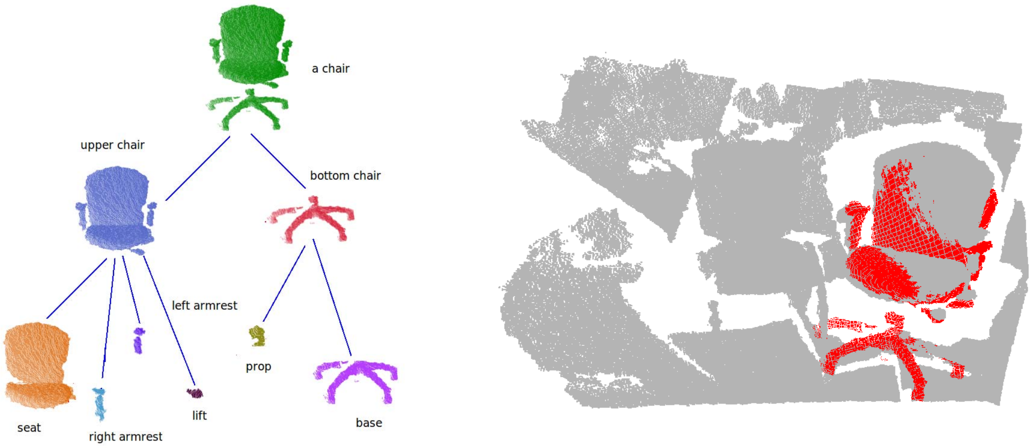


Figure 2.10: Left: a tree-structured model of a swivel chair; Right: pose estimation of the swivel chair within an office environment using a hierarchical conditional random field.

where $\pi(w_i, w_j)$ are *pairwise potential functions* associated with links between one part i and one of its child parts j. It can be any function that reflects the spatial constraint between i and j after their pose-corresponding transformation. The $\tau(w_k)$ are *unary potential functions* that reflect the likelihood of one part's pose given observation O. By applying an inference algorithm on the above conditional random field, all poses of different parts will reach a consensus, which is a globally optimal w^* based on the pose information of all levels. The performance of the model can be seen in one example in Figure 2.10.

2.3.3 Learning parametric abstractions of shapes using kernel methods

One of the central problems in capturing the environment by a robot is to interpret the objects observed. This interpretation can serve as a starting point to the potential activities. For example: can those objects be grasped, moved and reordered? The interpretation should not stick a label to those objects saying: this is a chair or that is a mug, but it should provide knowledge enough to act in a proper way. A mug or a glass, even a bottle, can be grasped in the same way, thus some common properties are really relevant among those objects, but others, e.g. colors, texture, some details of the shape can be ignored. Some parts, segments of the entire shape of the objects, carry activity-related properties that need to be captured.

The shapes of an object as an entity can not be directly observed by the known machine vision systems. Those systems can yield several different local feature items and the task is to build an abstract shape out of those features. Within that procedure we need to discover how those local items can relate to each other, what is the three dimensional graph connecting them, and recognize those items which can characterize the shape and separate them from those which can relate to something else, e.g. texture.

In learning shapes we assume features which can be characterized by two properties, a 3D position and an orientation, e.g. a surface segment, a patch, etc. These properties can be translated relatively easily into the properties of the potential activities. To collect this kind of features we need proper vision systems which can provide them with sufficient accuracy. Here we assume that these features are available for shape learning.

2.3.3.1 Shape model

We assume that the shape can be described as a manifold in the three dimensional space. This manifold, we might say surface, is an almost everywhere smooth one allowing to model edges and corners with high curvature, but otherwise it can be partitioned into relatively large connected smooth segments. This assumption expresses the need to eliminate irrelevant small details. Another requirement we should satisfy relates to the potential complexity of the shape, namely it can be a topologically higher order manifold with holes, with a mixture of segments with positive and negative curvature, and convex and concave parts.

To model a surface with complex structure and in the same time forcing a certain high level of smoothness we apply an infinite dimensional parametric representation exploiting the fact that a complex low dimensional manifold can be approximated by a hyperplane in a sufficiently high dimensional space.

The representation space we have chosen is an infinite dimensional Hilbert space of the square integrable functions. Within this space we can apply the probability density functions as features defined on the low dimensional 3D space to be modeled. This mathematical framework allows us to synthesize the probabilistic generative models and the robustness of the maximum margin based discriminative methods. Furthermore, the advantage of the kernel methods in expressing nonlinear relations can be exploited as well. The discrimination happens between the shape and the non-shape points, and the generative, density function based features provide certain local confidence measures on the shape approximation.

The shape modeling is considered as a machine learning procedure where the shape is extracted from local vision features. The learning task is to force a certain type of manifold to closely fit to the parameters, position and orientation, of the visual feature items, and in the same time it has to be as smooth, say simple, as possible which can be achieved via regularization constraining the complexity of the manifold applied.

The outcome of the learning method is an infinite dimensional vector, a combination of probability density functions. This kind of representation admits direct comparison of different objects to express their similarities and dissimilarities. This representation can be reused in other learning method to discover common parts within a given group of object, e.g. by applying Kernel Principal Component Analysis.

The derived shape models can be transformed by any affine transformation, e.g. translation or rotation, via acting on the parameters, expected values and/or covariance matrices, of the density functions used in the expression of the vectors describing the models.

The robot activities, e.g. grasping, can be modeled in a similar framework, thus both the shape models and the actions applied on those shapes as abstract vectors can be located in a common vector space. In this way potential connections between shapes and actions can be predicted for a new shape and for a new action. To do that the relationships between the novel items and the known ones needs to computed in the common vector space.

The details of the shape learning can be found in an attached technical report [SXP11].

2.3.3.2 Planned work

We are planning to incorporate features derived from edges only. Those features can be collected with less effort, require less sophisticated camera setups for capture, and can be computed significantly faster.

Another direction we started to work on is the incorporation of anisotropic kernels which emphasize the tangential directions against the radial ones on the surfaces. Since the anisotropic representation can express more flexible models, and it could lead to significantly smaller and simpler collection of features which can accelerate the processing.

20

Chapter 3

Visuo-haptic object representation

One key capability of autonomous robots consists in the ability of adapting their sensorimotor experience to new situations and tasks. For this purpose, processes for the exploration of unknown entities in the world are required to provide necessary perceptual abilities and generate suitable actions with the purpose of enhancing the knowledge of the system in an outside-in fashion. In task 2.1.2, we are focusing on the exploration of unknown objects by a humanoid robot with the goal to generate sensorimotor representations which are suitable for recognition, grasping, and manipulation.

In prior work, we investigated methods for the dexterous haptic exploration of unknown objects based on a potential field approach [10, 8, 9]). The feasibility of the approach has been demonstrated in the application of grasp affordances calculation based on the sensory data collected during the exploration process [11]. Since the proposed methods only generate sensory data based on the haptic domain, the exploration of integrated visuo-haptic object representations requires to extend the work in different directions. In the following, a brief summary of the conducted work toward integrated visuo-haptic object representations is outlined.

The haptic exploration in our previous work was restricted to single handed exploration tasks. While the single handed exploration constitutes a reasonable testbed for the potential field approach, it lacks applicability to many real world tasks. For the experiments, heavy objects had to be used or the objects had to be fixated in order to allow the integration of haptic data in a common reference frame. Further, due to kinematic restrictions and to possible collisions with the support plane the object could not be explored exhaustively. In order to overcome these limitations, we investigate the extension to bi-manual haptic exploration. Having grasped and fixated the object in one hand allows performing a more exhaustive exploration and overcomes the restriction to heavy or fixated objects. In order to grasp an object with little prior knowledge in an outside-in fashion, we developed methods for "blind" grasping to enable the bi-manual exploration process. The approach towards blind grasping is detailed in Section 3.1.

The exploration of visuo-haptic object representations requires to augment the haptic sensory data with visual information. Therefore, the binding problem across the visual and the haptic modality has to be addressed. In order to fuse haptic and visual sensory data, a common reference frame needs to be established. An approach allowing the establishment of such a common reference frame is proposed in Section 3.2. While seeking and successively keeping contact with an unknown object, the robot hand is localized visually. Using a model of the hand, the haptic sensory information relative to the tracked hand and the visual information from the camera images can be assigned to the same object thus yielding a visuo-haptic representation. The approach has been evaluated in a visuo-haptic object segmentation task.

3.1 Blind grasping for bi-manual visuo-haptic exploration

As explained in the last section, it is necessary to grasp an object for its bi-manual exploration. As the object is completely unknown before exploration, no model-based grasping techniques can be used. Therefore, we developed a system to grasp unknown objects from a table based on haptic sensor data. The search space where the unknown object(s) for exploration could be found is upper bounded by the reachable space of the robot. This rather big space needs to be restricted to a subspace where the objects can actually be found. In order to simultaneously reduce this space and provide the initial hypotheses (regions of interest) for the blind grasping, a simple and straightforward model-free vision method has been developed. The method consist of four stages:

1. Stereo images are captured by the calibrated stereo-rig and are used to extract a point-cloud of the scene.
2. Using the rigid transformation from the left camera to the robot platform, the point-cloud is mapped from perspective dependent camera coordinates to the invariant robot platform coordinates.
3. Using a bounding-box, it is possible to separate the points within in the volume of interest.
4. A kernel density estimation is used to determine the index of connectivity of points within the bounding box, in order to segment them into clusters, which are ranked by their size and high (z-coordinate). These clusters are hypotheses which can be used for model-free grasping towards bi-manual exploration, see Fig.3.1.

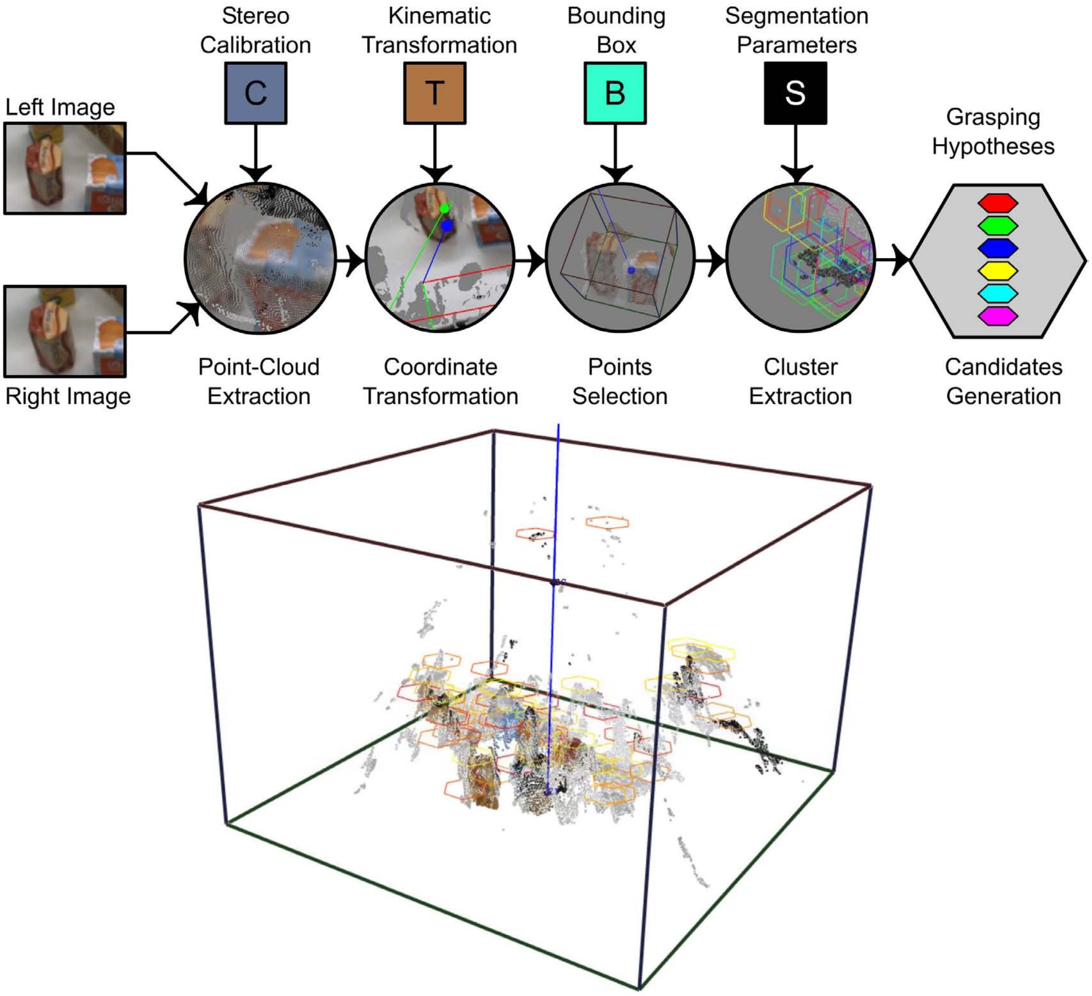


Figure 3.1: Schematic representation of the grasping hypotheses generation for the bi-manual visuo-haptic exploration.

Once a promising hypothesis for an object is found, the robot tries to reach this position. While reaching the robot uses the 6D force/torque sensor, which are mounted in the wrist, its tactile sensors in the finger tips and the encoder values of the hand to detect a contact with the object and estimate the position of the contact. The robot stops the movement upon contact and calculates a correction movement in direction of the estimated contact point. If another contact is detected during the correction movement, another correction movement is executed. The procedure is repeated until a contact in the palm of

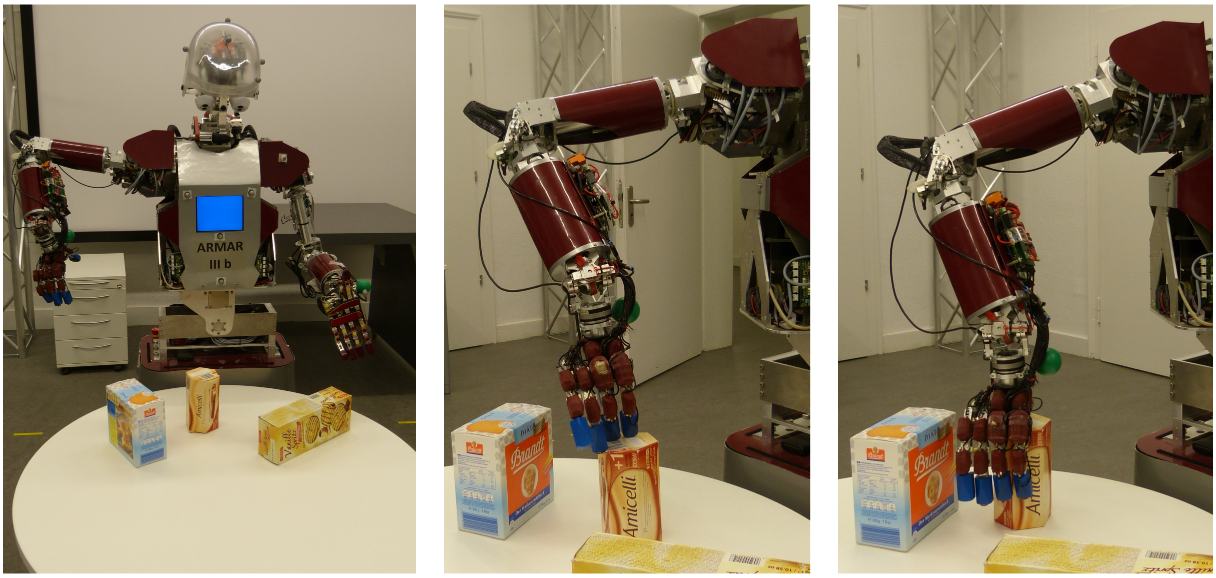


Figure 3.2: Different phases of the blind grasping: Generation of grasping hypothesis (left), contact detection with the object (center) and object grasping after performing corrective movements (right).

the hand is detected, an upper bound of correction movements is reached or the limit of the reachable space of the robot is reached. In the case of a contact on the palm, we assume that the object is inside the hand and the hand is closed trying to enclose the object. The procedure is shown in Fig. 3.2. A grasp stability check is performed using the position and tactile data. Only if the grasp is classified as stable, the object is lifted while continuously checking the grasp stability. In the case of reaching the upper bound of correction movements or the limit of the reachable space, the robot starts over with a new grasping hypothesis. The approach for grasp stability detection is presented in [SLP^{+}12]. This is based on temporal filtering of a support vector machine classifier output and estimates the stability continuously during the grasp attempt. Experimental evaluation on ARMAR-IIIb demonstrate that the estimation provides equal performance to the earlier approaches while reducing the time to reach a stable grasp significantly. Moreover, the results demonstrate for the first time that a learning based stability estimation can be used with a flexible, pneumatically actuated hand, in contrast to the rigid hands used in earlier works.

3.2 Visuo-haptic exploration for object segmentation

A prerequisite for performing visuo-haptic exploration is the establishment of a common reference frame which allows binding visual and haptic sensory data. For this purpose, the position of the tactile sensors of ARMAR-IIIb is registered with the visual coordinate frame by means of model-based tracking of the five-fingered hand. Using a kinematic model of the ARMAR-IIIb hand and the encoder readings from the finger joint sensors, the position of the finger tips are registered with the camera frame. Having established this common reference frame, visual as well as haptic sensory data can be fused in a single object representation.

The goal of the exploration process consisted in improving the figure-ground segmentation of objects in the camera image using haptic information. Figure-ground segmentation is one of the key prerequisite for the generation of object representations since it allows to bind sensory information to one single object. The figure-ground segmentation of an unknown object based on visual input alone is an ill-posed problem; if background and object exhibit a similar visual appearance, the pure visual segmentation will fail. The inclusion of haptic information allows performing a reliable figure-ground segmentation even in such situations, where object and background are visually ambiguous.

In order to verify the existence of an object at a spatial location using haptic sensory data, contact information is collected. Therefore, we use the force information on one fingertip to test spatial locations for object existence. The hand pose of the five-fingered hand used during exploration is illustrated

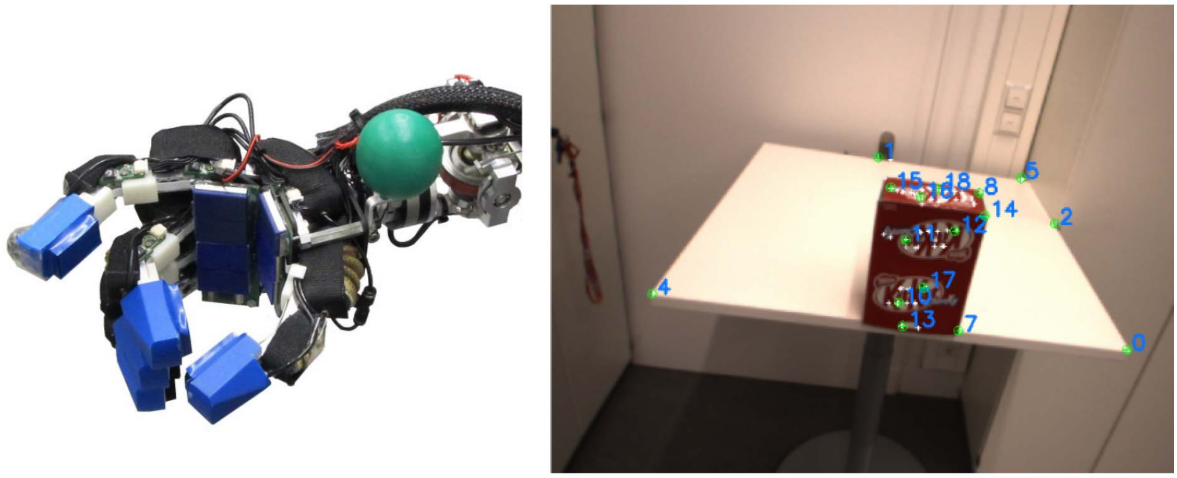


Figure 3.3: The contour of the visible surface of an unknown object is explored using the force measured on one fingertip. The configuration of the hand during exploration exposes this finger tip in order to establish and remain contact to the object (left). In order to identify potential candidates for exploration, unsupervised clustering is performed based on corner feature es. For each cluster center the exploration approach tries to establish contact with the object (right).

in Fig. 3.3, left. The force is derived using an approximated model of the pneumatic hand actuators according (see [12]).

In order to bind the contact information to the visual appearance of an unknown object, the exploration process establishes contact to the object perceived by the cameras. For this purpose, the scene is analyzed visually and from the visual information, hypotheses are generated in terms of potential 3D Cartesian target locations on the object. The exploration process then successively tries to establish contact at one of the potential target locations. For the calculation of potential targets, unsupervised clustering is performed on a set of texture features extracted from the current scene as illustrated in Fig. 3.3, right. Each cluster center is used as a potential target location. Based on the potential target location and the visual tracking of the five-fingered hand, a Visual Servoing approach has been implemented in order to establish the contact to the object. Using the texture features belonging to the hypothesis as reference the Visual Servoing successively reduces the distance between the fingertip and the selected potential target as illustrated in Fig. 3.4.

Once contact to the unknown object has been established, the exploration process starts to collect contact information and thus enrich the visually extracted information. The goal of the exploration consists in finding the extent of the object in the image plane of the camera. Therefore, a contour following approach has been proposed as exploration strategy. With a fixed orientation of the fingertip and a sensitive direction perpendicular to the image plane, the contour of the object is extracted. The exploration strategy is based on two exploration directions which are perpendicular to each other: the exploration direction and the testing direction. While keeping contact to the surface, the strategy moves discrete steps in the exploration direction resulting in discrete exploration points. If contact is lost, a new contour point is inserted, the fingertip is navigated to the last stable exploration point, and the exploration direction is turned counter-clockwise. On each exploration point, a movement in the test direction is performed in order to verify movement along the contour of the object. If the contact is not lost during the test movement, the exploration direction is turned clock-wise. Once the exploration process returns to the starting point of the exploration, the segmentation is finished. The contour of the object is recovered by connecting all exploration and contour points in their sequence of exploration.

Segmentation experiments were performed including simple box-shaped as well as more complex objects. More complex objects required a finer discretization of the exploration grid and thus resulted in a prolonged exploration process. For objects having a planar surface, the proposed exploration process allows reliable figure-ground segmentation. The inclusion of contact information allows to perform figure-ground segmentation of unknown objects even in scenes which contain visual ambiguities as can be seen in Figure 3.5. Also experiments on non-planar object have been performed. Therefore, the orientation of the sensitive direction has been adapted to the explored surface normal. Due to kinematic restrictions of the

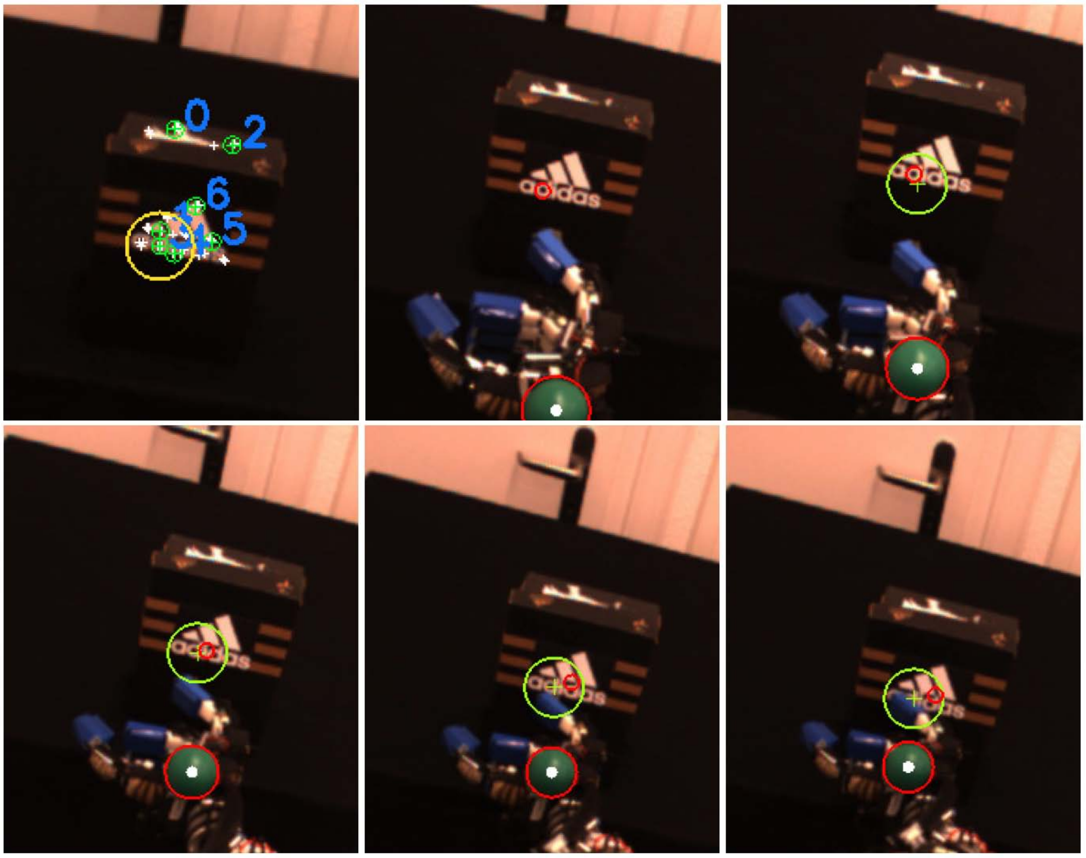


Figure 3.4: Using a Visual Servoing approach, a potential target on the surface of the object is approached in order to establish contact. Potential targets are defined by their cluster center and the associated texture features as resulting from unsupervised spatial clustering. The position of the fingertip is calculated using a model-based tracking of the five-fingered hand and the finger joint angle sensor readings. The texture features and the fingertip are tracked throughout the process in order to cope with head and hip movements of the robot.

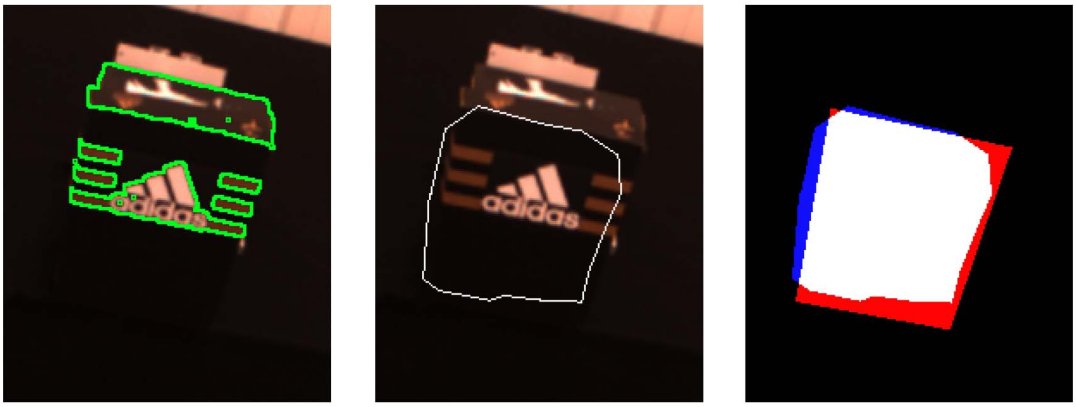


Figure 3.5: Results of the visuo-haptic exploration approach in figure-ground segmentation tasks. Due to ambiguities in the visual appearance of background and object the segmentation of the object based on visual data alone is affected (left). By enhancing the visual information using the proposed haptic exploration approach, the segmentation becomes more stable (middle). Compared to the ground truth, the visuo-haptic exploration approach is able to recover major parts of the contact surface (right).

ARMAR-III arm, the complete segmentation of such objects showed to be difficult. In such cases, the exploration would substantially benefit from a bi-manual approach.

Chapter 4

Reaching, grasping and manipulation

4.1 iCub architecture for reaching

4.1.1 Introduction

In this section we consider a solution to the problem of reaching for a visually identified target in a complex humanoid robot platform, considering both potential forceful interactions with objects or people and gross mistakes due to miscalibration of the controller parameters. Our reference platform is the iCub [23] – one of the platform of Xperience – a humanoid robot shaped as a three and half years old child. The iCub, by design, only uses "passive" sensors as for example cameras, gyroscopes, pressure, force and contact sensors, microphones and so forth. We excluded the use of lasers, sonars and other esoteric sensing modalities.

In this conditions and in an unstructured environment where human can freely move and work (our laboratory space in the daily use of the iCub), it is unlikely that the robot obtains an accurate model of the environment for accurate impact-free planning of movements. One common solution [32] is to control the robot mechanical impedance and, simultaneously, minimize impacts by using for example vision and trajectory planning. The possibility of impedance control lowers the requirements of vision and guarantees a certain degree of safety in case of contacts with the environment – though, strictly speaking, the robot can still be potentially dangerous and cause damage if it moves fast.

The control architecture described in this section is not very different in principle from a standard computed torque approach [31]. A first layer compensates for the dynamics and linearizes the system. Because of the communication bus of the iCub controllers, of bandwidth requirements, and implementation constraints, it operates in joint space. A second layer subsequently plans trajectories starting from a description of the target position in extrinsic space and merging joint limits, a secondary task specification, inverse kinematics and singularity avoidance. We show in the remainder of the section how this is implemented by mixing hand-coded models of the robot dynamics and kinematics together with machine learning.

Reaching and pointing is fundamental in learning about the environment enabling interaction with objects and their manipulation to achieve complex tasks. In this sense these are the basic building blocks of a complex cognitive architecture for the iCub and consequently for Xperience. This text and the following is larged imported from the ISRR 2011 invited paper where we summarized most of the work carried out on the iCub for the Xperience project.

4.1.2 Experimental platform: the iCub

The iCub is one of the results of the RobotCub project, an EU-funded endeavor to create a common platform for researchers interested in embodied artificial cognitive systems [2].

The initial specifications of the robot aimed at replicating the size of a three and a half years old child. In particular, it was required that the robot be capable of crawling on all fours and possess fine manipulation

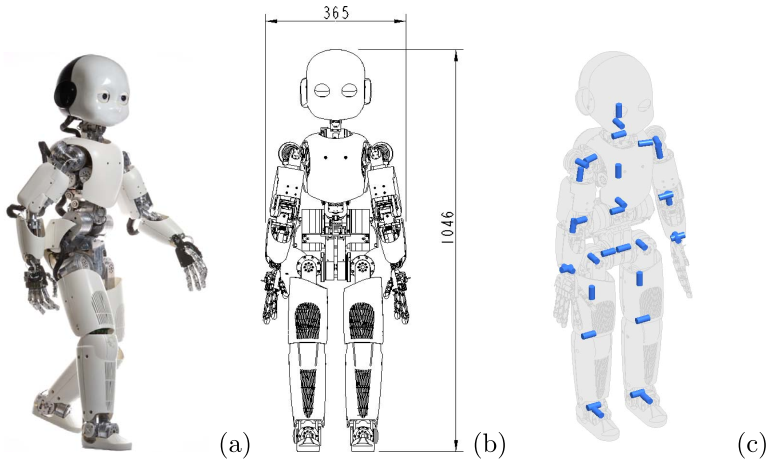


Figure 4.1: The iCub platform: panel (a) a picture of the latest realization of the iCub; panel (b) approximate dimensions height × width; and panel (c) the kinematic structure of the major joints.

abilities. For a motivation of why these features are important, the interested reader is referred to Metta et al. [22].

Dimensions, kinematic layout and ranges of movement were drafted by considering biomechanical models and anthropometric tables [34]. Rigid body simulations were used to determine the crucial kinematic features in order to perform the set of desired tasks and motions, i.e. reaching, crawling, etc. [35]. These simulations also provided joint torques requirements. Data were then used as a baseline performance indicator for the selection of the actuators. The final kinematic structure of the robot is shown in figure 4.1c. The iCub has 53 degrees of freedom (DoF). Its kinematics has several special features which are rarely found in other humanoid robots: e.g. the waist has three DoF which considerably increase the robot's mobility; the three DoF shoulder joint is constructed to have its axes of rotation always intersecting at one point.

To match the torque requirements we employed rotary electric motors coupled with speed reducers. We found this to be the most suitable choice in terms of robustness and reliability. Motor groups with various characteristics were developed (e.g. 40Nm, 20Nm and 11Nm) for different placements into the iCub. We used the Kollmorgen-DanaherMotion RBE type brushless frameless motor (BLM) and a CSD frameless Harmonic Drive as speed reducer. The use of frameless components allowed further optimization of space and reduced weight. Smaller motors for moving the fingers, eyes and neck are from Fulhaber in various sizes and reduction gear ratios.

Cable drives were used almost everywhere on the iCub. Most joints have relocated motors as for example in the hand, shoulder (besides one joint), elbow, waist and legs (apart from two joints). Cable drives are efficient and almost mandatory in order to optimize the motor locations and the overall "shape" of the robot. All joints in the hand are cable driven. The hand of the iCub has 20 joints which are moved by only 9 motors: this implies that some of the joints are under-actuated and their movement is obtained by means of the cable couplings. Similarly to the human body most of the hand actuation is in the forearm subsection. The head is another particular component of the iCub enabling independent vergence movements supported by a three DoF neck for a total of six DoF.

By design we decided to only use "passive sensors" and in particular cameras, microphones, gyroscopes and accelerometers, force/torque (FTS) and tactile sensors as well as the traditional motor encoders. Of special relevance is the sensorized skin which is not easily found in other platforms as well as the force/torque sensors that are used for force/impedance control (see later). No active sensing is provided as for example lasers, structured light projectors, and so forth.

The iCub mounts custom-designed electronics which consists of programmable controller cards, amplifiers, DACs and digital I/O cards. This ecosystem of microcontroller cards relies on multiple CAN bus lines (up to 10) for communication and synchronization and then connects with a cluster of external machines via a Gbit/s Ethernet network. Data are acquired and synchronized (and timestamped) before being made available on the network. We designed the software middleware that supports data acquisition and control of the robot as well as all the firmware that operates on the microcontrollers which eventually drive each single transistor that moves the motors.

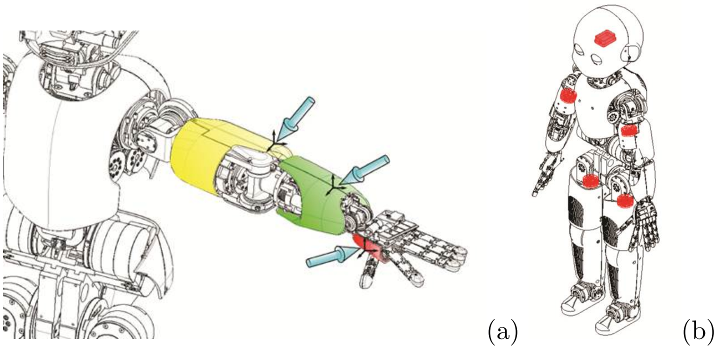


Figure 4.2: In (a) a typical interaction of the iCub arm with the environment exemplified here with a number of wrenches at different locations and in (b) the location of the four FTSs of the iCub in the upper part of the limbs (proximal with respect to the reference frame of the robot kinematic chains) and of the inertial sensors mounted in the head.

The software middleware is called YARP [15]. YARP is a thin library that enables multi-platform and multi-IDE development and collaboration by providing a layer that shields the user from the quirks of the underlying operating system and robot hardware controllers. The complete design of the iCub (drawings, schematics, specifications) and its software (both middleware and controllers) is distributed according to the GPL or the LGPL licenses. In Xperience we decided to interface YARP with other middlewares as for example the widely employed ROS (www.ros.org) and ICE (http://www.zeroc.com/).

4.1.3 Dynamics

The first layer of the proposed architecture is based on computation of the body dynamics and implements joint position & velocity control on top of joint-level impedance. In the simplest possible version, the controller cards implement a 1ms feedback loop relying on the error e defined as:

$$e = \tau - \tau_d \ , \tag{4.1}$$

where τ is the vector of joint torques and τ_d its desired value. We do not know τ directly on the iCub but we have access to estimates through the force/torque sensors (FTSs). They are mounted as indicated in figure 4.2 in the upper part of the limbs and can therefore be used to detect wrenches at any location in the iCub limbs and not only at the end-effector as it is more typical for industrial manipulators.

We show that τ can be estimated from the FTS measurements of each limb (equations repeat identical for each limb). Let's indicate with w_s the wrench measured by the FTS and assume that it is due to an actual external wrench at a known location (e.g. at the end-effector) which we call w_e. We can estimate w_e by propagating the measurement on the kinematic chain of the limb (changing coordinates):

$$\hat{w}_e = \begin{bmatrix} I & 0 \\ -[\bar{r}_{se}]_\times & I \end{bmatrix} \cdot (w_s - w_i) \ , \tag{4.2}$$

with $[\bar{r}_{se}]_\times$ the skew-symmetric matrix representing the cross product with the vector $\bar{r}_{se}$, $\hat{w}_e$ the estimate of w_e, and w_i the internal wrench (due to internal forces and moments). Note that $[\bar{r}_{se}]_\times$ is a function of q, the vector of joint angles. w_i can be estimated from the dynamics of the limb (either with the Lagrange or Newton-Euler formulation). To estimate τ_e we only need to project $\hat{w}_e$ to the joint torques using the transposed Jacobian, i.e.:

$$\hat{\tau}_e = J^T(q) \cdot \hat{w}_e \ . \tag{4.3}$$

We can then use this estimate in a control loop by defining the torque error e as:

$$e = \hat{\tau}_e - \tau_d \ , \tag{4.4}$$

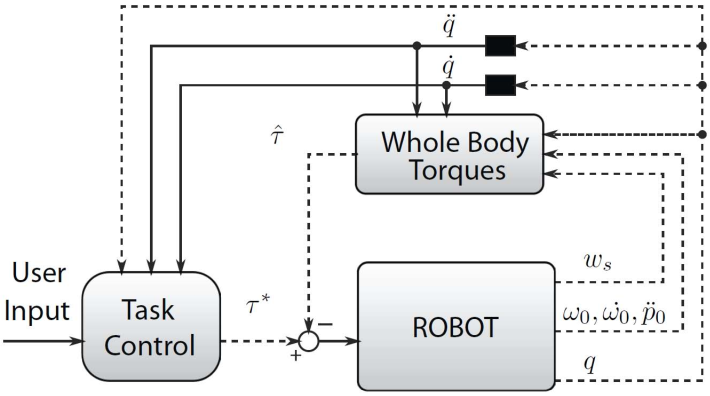


Figure 4.3: The torque controller of the iCub. See text for details.

where $\hat{\tau}_e$ is an estimate of τ regulated by a PID controller of the form:

$$u = k_p \cdot e + k_d \cdot \dot{e} + k_i \cdot \int e \ , \tag{4.5}$$

where k_p, k_d and k_i are the usual PID gains and u the amplifier output (the PWM duty cycle which determines the equivalent applied voltage at the motor). Similarly we can build an impedance controller in joint space by making τ_d of the form:

$$\tau_d = K \cdot (q - q_d) + D \cdot (\dot{q} - \dot{q}_d) \ , \tag{4.6}$$

which can be implemented at the controller card level if K and D are diagonal matrices. Furthermore, we can command velocity by making:

$$q_d(t) = q_d(t - \delta t) + \dot{q}_d(t)\delta t \ , \tag{4.7}$$

with δt the control cycle interval (1ms in our case). This latter modality is useful when generating whole trajectories incrementally. The actual computation of the dynamics and kinematics is based on a graph representation which we detail in the following.

We start by considering an open (single or multiple branches) kinematic chain with n DoF composed of $n + 1$ links. Adopting the Denavit-Hartenberg notation [31], we define a set of reference frames $\langle 0 \rangle$, $\langle 1 \rangle$, ..., $\langle n \rangle$, attached at each link. The i^{th} link of the chain is described by a vertex v_i (sometimes called node), usually represented by the symbol $\textcircled{$i$}$. A hinge joint between the link i and the link j (i.e. a rotational joint) is represented by an oriented edge $e_{i,j}$ connecting v_i with v_j: $\textcircled{i} \rightarrow \textcircled{j}$. In a n DoF open chain, each vertex (except for the initial and terminal, v_0 and v_n respectively) has two edges. Therefore, the graph representation of the n-link chain is an oriented sequence of nodes v_i, connected by edges $e_{i-1,i}$. The orientation of the edges can be either chosen arbitrarily (it will be clear later on that the orientation simply induces a convention) or it can follow from the exploration of the kinematic tree according to the *regular numbering scheme* [14], which induces a parent-child relationship such that each node has a unique input edge and multiple output edges. We further follow the classical Denavit-Hartenberg notation, we assume that each joint has an associated reference frame with the z-axis aligned with the rotation axis; this frame will be denoted $\langle e_{i,j} \rangle$. In kinematics, an edge $e_{i,j}$ from v_i to v_j represents the fact that $\langle e_{i,j} \rangle$ is fixed in the i^{th} link. In dynamics, $e_{i,j}$ represents the fact that the dynamic equations will compute (and make use of) $w_{i,j}$, i.e. the wrench that the i^{th} link exerts on the j^{th} link, and not the equal and opposite reaction $-w_{i,j}$, i.e. the wrench that the j^{th} link exerts on the i^{th} link. In order to simplify the computations of the inverse dynamics on the graph, kinematic and dynamic measurements have been explicitly represented. Specifically, the graph representation has been enhanced with a new set of graphical symbols: a triangle to represent kinematic quantities (i.e. velocities and acceleration of links - ω, $\dot{\omega}$, $\dot{p}$, $\ddot{p}$), and a rhombus for wrenches (i.e. force sensors measurements on a link - f, μ). Moreover

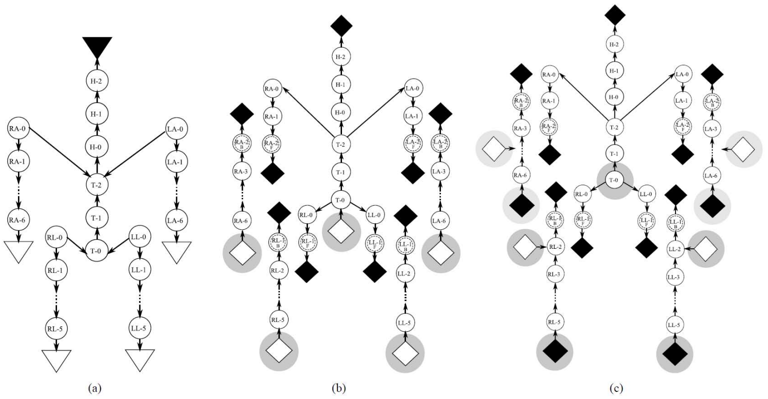


Figure 4.4: Representation of iCub's kinematic and dynamic graph. In (a): iCub's kinematics. The inertial sensor measure (▼) is the unique source of kinematic information for the whole branched system. (b): iCub's dynamics when the robot is standing on the mainstay and moving freely in space. Given the four FTSs, the main graph is cut by the four links hosting the sensors, and a total of five sub-graphs are finally generated. The unknowns are the external wrenches at the end-effectors: if the robot does not collide with the environment, they are zero, whereas if a collision happens, then an external wrench arises. The displacement between the expected and the estimated wrenches allows detecting contacts with the environment under the hypothesis that interactions can only occur at the end-effectors. The external wrench on top of the head is assumed to be null. Notice that the mainstay is represented by a unknown wrench ◇. (c): iCub's dynamics when the robot is crawling (four points of contact with the ground). As in the previous case, five sub-graphs are generated after the insertion of the four FTSs measurements, but unlike the free-standing case, here the mainstay wrench is removed, being the iCub on the floor. Specific locations for the contacts with the environment are given as part of the task: the unknown external wrenches (◇) are placed at wrists and knees, while wrenches at the feet and palms are assumed known and null (▼). Interestingly, while moving on the floor the contact with the upper part could be varying (e.g. wrists, palms, elbows), so the unknown wrenches could be placed in different locations than the ones shown in the graph.

these symbols have been further divided into known quantities to represent sensors measurements, and unknown to indicate the quantities to be computed, as in the following:

- ∇: unknown kinematic information
- ▼: known (e.g., measured) kinematic information
- ◇: unknown dynamic information
- ⧫: known (e.g., measured) dynamic information

In general, kinematic variables can be measured by means of gyroscopes, accelerometers, or simply inertial sensors. When attached on link i^{th}, these sensors provide angular and linear velocities and accelerations (ω, $\dot{\omega}$, $\dot{p}$ and $\ddot{p}$) at the specific location where the sensor is located. We can represent these measurement in the graph with a *black triangle* (▼) and an additional edge from the proper link where the sensor is attached to the triangle. As usual, the edge has an associated reference frame, in this case corresponding to the reference frame of the sensor. An unknown kinematic variable is represented by a *white triangle* (∇) with an associated edge going from the link (where the unknown kinematic variable is attached) to the triangle. Similarly, we introduce two new types of nodes with a rhomboidal shape: *black rhombus* (⧫) to represent known (i.e. measured) wrenches, *white rhombus* (◇) to represent unknown wrenches which need to be computed. The reference frame associated to the edge will be the location of the applied or unknown wrench. The complete graph for the iCub is shown in figure 4.4.

From the graph structure, we can define the update rule that brings information across edges and by traversing the graph we therefore compute either dynamical or kinematic unknowns (◇ and ∇ respectively). For kinematic quantities this is:

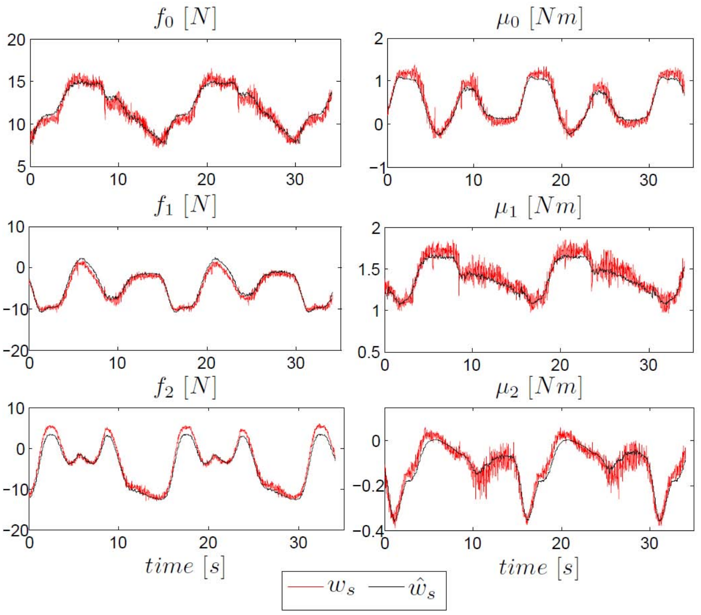


Figure 4.5: Comparison between the wrench measured by the FT sensor and that predicted by the model, during a generic contact-free movement of the left arm. The three plots on the left are forces expressed in [N]; the three rightmost plots are the moments in [Nm].

$$\begin{aligned} \omega_{i+1} &= \omega_i + \dot{\theta}_{i+1} z_i \ , \\ \dot{\omega}_{i+1} &= \dot{\omega}_i + \ddot{\theta}_{i+1} z_i + \dot{\theta}_{i+1} \omega_i \times z_i \ , \\ \ddot{p}_{i+1} &= \ddot{p}_i + \dot{\omega}_i \times r_{i,i+1} + \omega_{i+1} \times (\omega_{i+1} \times r_{i,i+1}) \ , \end{aligned} \tag{4.8}$$

where z_i is the z-axis of $\langle i \rangle$, i.e. we propagate information from the base to the end-effector visiting all nodes and moving from one node to the next following the edges. The internal dynamics of the manipulator can be studied as well: if the dynamical parameters of the system are known (mass m_i, inertia I_i, center of mass C_i), then we can propagate knowledge of wrenches applied to e.g. the end-effector (f_{n+1} and μ_{n+1}) to the base frame of the manipulator so as to retrieve forces and moments f_i, μ_i:

$$\begin{aligned} f_i &= f_{i+1} + m_i \ddot{p}_{C_i} \ , \\ \mu_i &= \mu_{i+1} - f_i \times r_{i-1,C_i} + f_{i+1} \times r_{r_i,C_i} + I_i \dot{\omega}_i + \omega_i \times (I_i \omega_i) \ , \end{aligned} \tag{4.9}$$

where:

$$\ddot{p}_{C_i} = \ddot{p}_i + \dot{\omega}_i \times r_{i,C_i} + \omega_i \times (\omega_i \times r_{i,C_i}) \ , \tag{4.10}$$

noting that these are the classical recursive Newton-Euler equations. Knowledge of wrenches enables the computation of w_i as needed in equation 4.2 or the corresponding joint torques from $\tau_i = \mu_i^T z_{i-1}$.

4.1.3.1 Validation and further improvements

In order to validate computation of the dynamics, we compared measurements from the FTSs with their model-based prediction. The wrenches w_s from the four six-axes FTSs embedded in the limbs are compared with the analogous quantities $\hat{w}_s$ predicted by the dynamical model, during unconstrained movements (i.e. null external wrenches). Kinematic and dynamic parameters are retrieved from the CAD model of the robot. Sensor measurements w_s can be predicted assuming known wrenches at the limbs extremities (hands or feet) and then propagating forces up to the sensors. In this case, null wrenches are assumed, because of the absence of contact with the environment. Table 4.1 summarizes the statistics of the errors ($w_s - \hat{w}_s$) for each limb during a given, periodic sequence of movements, with the robot supported by a rigid metallic mainstay, and with the limbs moving freely without self collision or contact

Table 4.1: Error in predicting FT sensor measurement (see text for details).

	ϵ_{f0}	ϵ_{f1}	ϵ_{f2}	$\epsilon_{\mu 0}$	$\epsilon_{\mu 1}$	$\epsilon_{\mu 2}$
$\bar{\epsilon}$	-0.3157	-0.5209	0.7723	-0.0252	0.0582	0.0197
σ_ϵ	0.5845	0.7156	0.7550	0.0882	0.0688	0.0364
right arm: $\epsilon \equiv \hat{w}_{s,RA} - w_{s,RA}$						
$\bar{\epsilon}$	-0.0908	-0.4811	0.8699	0.0436	0.0382	0.0030
σ_ϵ	0.5742	0.6677	0.7920	0.1048	0.0702	0.0332
left arm: $\epsilon \equiv \hat{w}_{s,LA} - w_{s,LA}$						
$\bar{\epsilon}$	-1.6678	3.4476	-1.5505	0.4050	-0.7340	0.0171
σ_ϵ	3.3146	2.7039	1.7996	0.3423	0.7141	0.0771
right leg: $\epsilon \equiv \hat{w}_{s,RL} - w_{s,RL}$						
$\bar{\epsilon}$	0.2941	-5.1476	-1.9459	-0.3084	-0.8399	0.0270
σ_ϵ	1.8031	1.8327	2.3490	0.3365	0.8348	0.0498

left leg: $\epsilon \equiv \hat{w}_{s,LL} - w_{s,LL}$
SI units: $f : [N]$, $\mu : [Nm]$

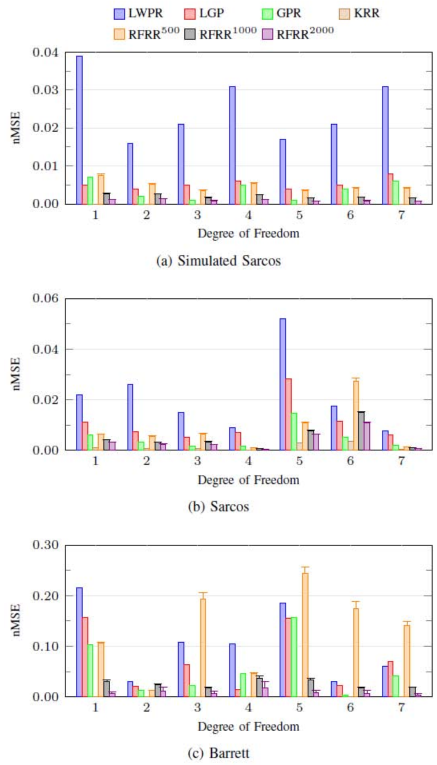
with the environment. Table 4.1 shows the mean and the standard deviation of the errors between measured and predicted sensor wrench during movement. Figure 4.5 shows a comparison between w_s and $\hat{w}_s$ for the left arm (without loss of generality, all limbs show similar results).

Subsequently we investigated methods to improve the estimates of the robot dynamics. In another set of experiments we thus compared various non-parametric learning methods with the rigid body model just presented. We refer the interested reader to Gijsberts et al. [17]. We report here only the main findings. The task of learning here is the estimation of the wrenches due to the internal dynamics (w_i) given the FTS readings (w_s) and the robot configuration (q, $\dot{q}$, $\ddot{q}$); we do not take into account inertial information.

We compared various methods from the literature as for example the widely used Local Weighted Projection Regression (LWPR), the Local Gaussian Process (LGP) and Gaussian Process Regression (GPR) as presented by Nguyen-Tuong et al. [25] with an incremental version of Kernel Ridge Regression (also known as Sparse Spectrum Gaussian Process) with the aim of maintaining eventually an incremental open-ended learner updating the estimation of the robot dynamics on-line. Our incremental method relies on an approximation of the kernel (see [30]) based on a random sampling of its Fourier spectrum. The more random features, the better the approximation. We considered approximations with 500, 1000, and 2000 features. In the following we call KRR the plain kernel ridge regression method and RFRR^D the random feature version for D features. Various datasets (e.g. Barret, Sarcos) were used from the literature (for comparison [25]) before applying the method to the iCub.

The results in figure 4.6 show that KRR often outperforms GPR by a significant margin, even though both methods have identical formulations for the predictive mean and KRR hyperparameters were optimized using GPR. These deviations indicate that different hyperparameter configurations were used in both experiments. This is a common problem with GPR in comparative studies: the marginal likelihood is non-convex and its optimization often results in a local optimum that depends on the initial configuration. Hence, we have to be cautious when interpreting the comparative results on these datasets with respect to generalization performance. The comparison between KRR and RFRR, trained using identical hyperparameters, remains valid and gives an indication of the approximation quality of RFRR. As expected, the performance of RFRR steadily improves as the number of random features increases. Furthermore, RFRR^{1000} is often sufficient to obtain satisfactory predictions on all datasets. RFRR^{500}, on the other hand, performs poorly on the Barrett dataset, despite using distinct hyperparameter configurations for each degree of freedom. In this case, RFRR^{1000} with a shared hyperparameter configuration is more accurate and requires overall less time for prediction.

Figure 4.7 shows how the average nMSE develops as test samples are predicted in sequential order using either KRR or RFRR. RFRR requires between 5000 and 10000 samples to achieve performance comparable to KRR. The performance of KRR, on the other hand, decreases over time. In particular on



(a) Simulated Sarcos

(b) Sarcos

(c) Barrett

Figure 4.6: Prediction error per degree of freedom for the (a) Simulated Sarcos, (b) Sarcos, and (c) Barrett datasets. The results for LWPR, GPR, and LGP are taken from Nguyen-Tuong et al. [25]. The mean error over 25 runs is reported for RFRR with $D \in 500, 1000, 2000$, whereas error bars mark a distance of one standard deviation. Note that in some cases the prediction errors for KRR are very close to zero and therefore barely noticeable.

the iCub dataset it suffers a number of large errors, causing the average nMSE to show sudden jumps. This is a direct consequence of the unavoidable fact that training and test samples are not guaranteed to be drawn from the same distribution. Incremental RFRR, on the other hand, is largely unaffected by these changes and demonstrates stable predictive performance. This is not surprising, as RFRR is incremental and thus (1) it is able to adapt to changing conditions, and (2) it eventually has trained on significantly more samples than KRR. Furthermore, figure 4.7 shows that 200 random features are sufficient to achieve satisfactory performance on either dataset. In this case, model updates of RFRR require only $400\mu s$, as compared to $2ms$ and $7ms$ when using 500 or 1000 random features, respectively. These timing figures make incremental RFRR suitable for high frequency loops as needed in robot control tasks.

In conclusion, this shows that for a relatively complex robot like the iCub, good estimation of the internal dynamics is possible and that a combination of non-parametric and parametric methods can provide simultaneously good generalization performance, fast and incremental learning. Not surprisingly, lower errors are obtained with learning. In the next section we see how to build on this controller to reach for visually identified targets.

4.1.4 Kinematics

We consider the general problem of computing the value of joint angles q_d in order to reach a given position in space $x_d \in \mathbb{R}^3$ and orientation $\alpha_d \in \mathbb{R}^4$ of the end-effector (where α_d is a representation of rotation in axis/angle notation). Note that q_d can be directly connected to the input of the impedance controller described in section 4.1.3. It is desired that the computed solution satisfies a set of additional constraints expressed as generic inequalities – we see later the reason for constraining the solution of the optimization problem. This can be stated as follows:

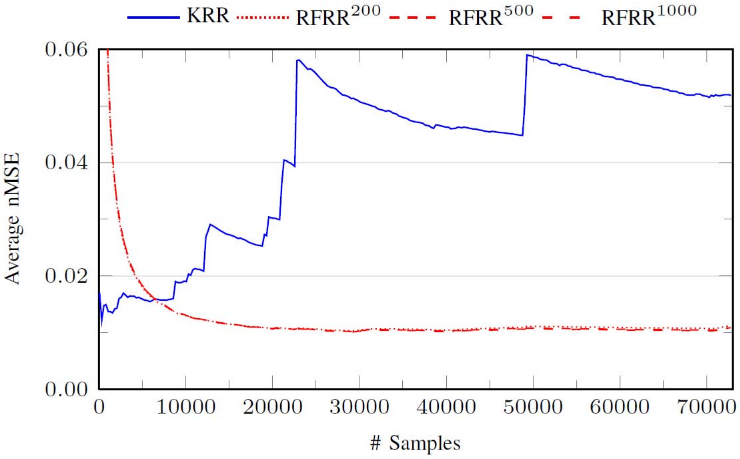


Figure 4.7: Average prediction error with respect to the number of test samples of KRR and incremental RFRR with $D \in 200, 500, 1000$ on the iCub dataset. The error is measured as the nMSE averaged over the force and torque output components. The standard deviation over 25 runs of RFRR is negligible in all cases, for clarity we report only the mean without error bars.

$$
\begin{aligned}
&q_d = \text{argmin}_{q \in \mathbb{R}^n} \left(\|\alpha_d - K_\alpha(q)\|^2 + \beta (q_{rest} - q)^T W (q_{rest} - q) \right) , \\
&s.t. \begin{cases} \|x_d - K_x(q)\|^2 < \epsilon \\ q_L < q < q_U \end{cases} ,
\end{aligned}
\tag{4.11}
$$

where K_x and K_α are the forward kinematic functions for the position and orientation of the end-effector for a given configuration q; q_{rest} is a preferred joint configuration, W is a diagonal weighting matrix, β a positive scalar weighting the influence of the terms in the optimization and ϵ a parameter for tuning the precision of the movement. Typically $\beta < 1$ and $\epsilon \in [10^{-5}, 10^{-4}]$. The solution to equation 4.11 has to satisfy the set of additional constraints of joint limits $q_L < q < q_U$ with q_L, q_U the lower and upper bounds respectively. In the case of the iCub, we solved this problem for ten DoF – seven of the arm and three of the waist and we determined the value of q_{rest} so that the waist is as upright as possible. The left and right arm can be both controlled by switching from one or the other kinematic chain (e.g. as a function of the distance to the target).

We used an interior point optimization technique to solve the problem in 4.11. In particular we used IpOpt [36], a public domain software package designed for large-scale nonlinear optimization. This approach has the following advantages:

1. Quick convergence. IpOpt is reliable and fast enough to be employed in control loops at reasonable rates (tens of milliseconds), as e.g. compared to more traditional iterative methods such as the Cyclic Coordinate Descent (CCD) adopted in [18];

2. Scalability. The intrinsic capability of the optimizer to treat nonlinear problems in any arbitrary number of variables is exploited to make the controller structure easily scalable with the size of the joint space. For example, it is possible to change at run time from the control of the 7-DoF iCub arm to the complete 10-DoF structure inclusive of the waist or to any combination of the joints depending on the task;

3. Automatic handling of singularities and joint limits. This technique automatically deals with singularities in the arm Jacobian and joint limits, and can find solutions in virtually any working conditions;

4. Tasks hierarchy. The task is split in two subtasks: the control of the orientation and the control of the position of the end-effector. Different priorities can be assigned to the subtasks. In our case the control of position has higher priority with respect to orientation (the former is handled as a nonlinear constraint and thus is evaluated before the cost);

5. Description of complex constraints. It is easy to add new constraints as linear and/or nonlinear inequalities either in task or joint space. In the case of the iCub, for instance, we added a set

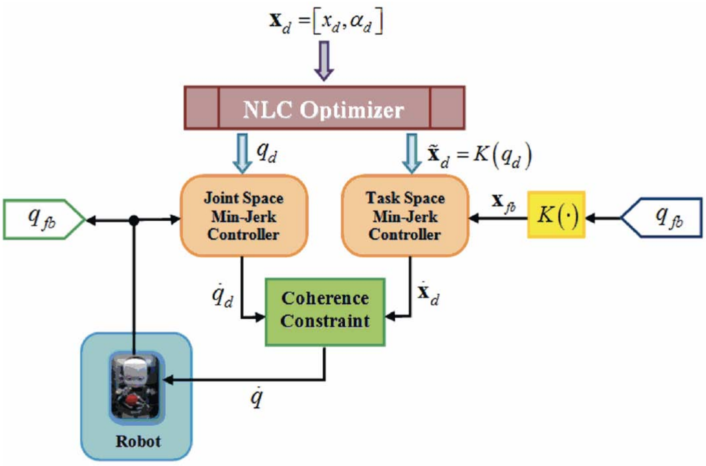


Figure 4.8: The multi-referential scheme for trajectory generation. K is the forward kinematics map; q_{fb} is the vector of encoder signals.

of constraints that avoid reaching the limits of the tendons that actuate the three joints of the shoulder.

Once q_d is determined as described above, there is still the problem of generating a trajectory from the current robot configuration q to q_d. Simultaneously, we would like to impose suitable smoothness constraints to the trajectory. This has been obtained by using the Multi-Referential Dynamical Systems approach [18], whereby two dynamical controllers, one in joint space and another in task space, evolve concurrently (figure 4.8). The coherence constraint, that is $\dot{x} = J\dot{q}$, with J the Jacobian of the kinematics map, guarantees that at each instant of time the trajectory is meaningful. This is enforced by using the Lagrangian multipliers method and can be tuned to modulate the relative influence of each controller (i.e. to avoid joint angles limits). The advantage of such a redundant representation includes the management of the singularities while maintaining a quasi-straight trajectory profile of the end-effector in the task space – reproducing a human-like behavior [4].

Differently from the work of Hersch and Billard, we designed a feedback trajectory generator instead of the VITE (Vector- Integration-To-Endpoint) method used in open loop. A complete discussion of the rationale of the modifications to the trajectory generation is outside the scope of this report; the interested reader is referred to Pattacini et al. [26]. Reasons to prefer a feedback formulation include the possibility of smoothly connecting multiple pieces of trajectories and correcting on line for accumulation of errors due to the enforcement of the constraints of the multi-referential method.

4.1.4.1 Validation and further improvements

As earlier for the dynamics, we compared our method with other methods from the literature. The comparison with the method of Hersch et al. [18] was almost immediate since the work was developed on the iCub. This provides the multi-referential approach together with the VITE trajectory generation at no cost. Additionally, we included in the assessment another controller representing a more conventional strategy that uses the Damped Least-Squares (DLS) rule [13] coupled with a secondary task that comprises the joints angles limits by means of the gradient projection method [20]. This solution employs the third-party package Orocos [1], a tool for robot control that implements the DLS approach and whose public availability and compliance with real-time constraints justified its adoption as one of the reference controllers.

In the first experiment we put to test the three selected schemes in a point-to-point motion task wherein the iCub arm was actuated in the "7-DoF mode" and where the end-effector was controlled both in position and orientation. Results show that paths produced by our controller and by the DLS-based system are well restricted in narrow tubes of confidence intervals and are quite repeatable; conversely the VITE is affected by a much higher variability. Figure 4.9 highlights results for a set of 10 trials of

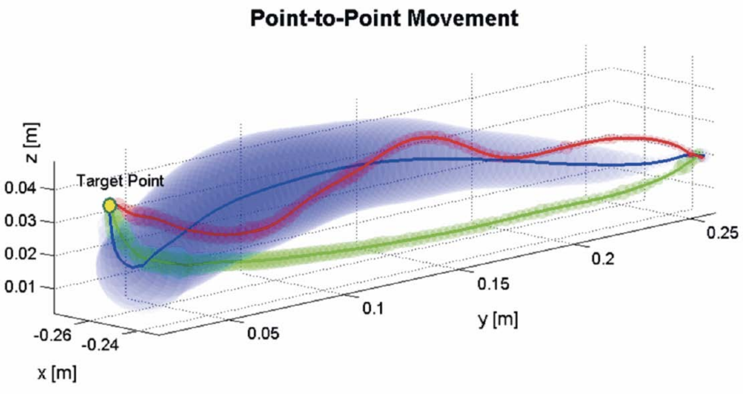


Figure 4.9: Point-to-point Cartesian trajectories executed by the three controllers: the VITE-based method produces on average the blue line, the minimum-jerk controller result is in green, the DLS system using Orocos in red. Bands containing all the measured paths within a confidential interval of 95% are drawn in corresponding colors. Controllers settings are: $T = 2.0s$ for the minimum-jerk system, $\alpha = 0.008$, $\beta = 0.002$, $K_P = 3$ for the VITE (see [18] for the meaning of the parameters), and $\mu = 10^{-5}$ for the damping factor of the DLS algorithm.

Table 4.2: Mean errors along with the confidence levels at 95% computed when the target is attained. An average measure of the variability of executed path is also given for the three controllers.

Controller	Position error	Orientation error	Mean radius of the trajectory band
VITE	$1.3 \pm 1.4 \cdot 10^{-3} mm$	$0.041 \pm 0.05 rad$	$10 \pm 10.8 mm$
Min-jerk	$3.0 \pm 1.3 \cdot 10^{-3} mm$	$0.048 \pm 0.008 rad$	$2.5 \pm 1.5 mm$
DLS	$1.3 \pm 1.4 \cdot 10^{-3} mm$	$0.016 \pm 0.028 rad$	$2.0 \pm 1.36 mm$

a typical reaching task where the right hand is moved from a rest position to a location in front of the iCub with the palm directed downward.

Table 4.2 summarizes the measured in-target errors for the three cases: all the controllers behave satisfactory, but the DLS achieves lower errors because operates continuously on the current distance from the target x_d , being virtually capable of canceling it at infinite time. On the contrary, strategies based on the interaction with an external solver bind the controller module to close the loop on an approximation $\tilde{x}_d$ of the real target that is determined by the optimization tolerances as in 4.11.

Additional experiments tend to favor our method. For example, measuring the jerk of the resulting trajectory shows a gain of our method by 43% from the VITE and of about 69% from DLS. This turns out to be crucial for more complicated trajectories when speed factors make the minimum jerk controller even more advantageous.

Further improvements can be made on the quality of the inverse kinematic results by means of machine learning. As for the dynamics, we initially estimated the function K from the CAD models of the iCub. This is a good initial guess in need of refinement. The goal here is therefore to design a procedure that allows enforcing eye-hand coordination such that, whenever the robot reliably localizes a target in both cameras, it can also reach it. Here we further simplified the problem (from the visual point of view) and decided to learn only the position of the end-effector (x, y, z) since the orientation of the hand in the image is difficult to detect reliably. For this problem, the input space is defined by the position of the hand (or the target) in the two cameras (u_l, v_l) and (u_r, v_r) with respect to the current head configuration.

To sum up, having defined the input and the output space, the map M that is to be learned is:

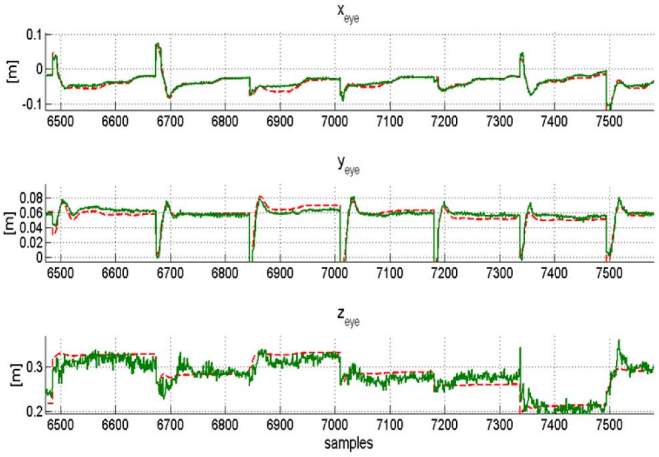


Figure 4.10: The desired target (dashed red) and the corresponding outputs of the neural network (green) for the three Cartesian coordinates in the head centered frame.

$$(x, y, z)_H = M(u_l, v_l, u_r, v_r, T, V_s, V_g) \ , \tag{4.12}$$

where $(u_l, v_l, u_r, v_r) \in \mathbb{R}^4$ represent the visual input of the position of the hand in the iCub cameras, whereas $(T, V_s, V_g) \in \mathbb{R}^3$ accounts for the proprioceptive part of the input designating the tilt, the pan and the vergence of the eyes; finally, $(x, y, z)_H \in \mathbb{R}^3$ is the Cartesian position of the hand expressed in the head-centered frame.

This map can be learned by a regression method if enough training samples are available and these can be in turn collected if we can measure (u_l, v_l, u_r, v_r) by means of vision (see section 4.1.5). Some preliminary results by using a sigmoidal neural network from Matlab (Neural Network Toolbox) trained with backpropagation can be seen in figure 4.10. The training phase is carried out off-line. The neural network consists of 7 nodes in the linear input layer, 50 nodes for the hidden layer implemented with the ordinary hyperbolic tangent function and 3 nodes in the linear output layer: an overall number of 15000 samples has been employed for training and validation, whereas 5000 samples have been used for testing. The neural network provides a very good estimation of M as demonstrated by the testing phase. Notably, as expected, the z component estimation is the most affected by noise since it accounts principally for the distance of the hand from the head, a value that is not directly measured by the cameras but only indirectly from binocular disparity. The inspection of the mean and standard deviation supports this claim, i.e. mean error $0.00031m$ and standard deviation of $0.0055m$ for the x and y components and about twice as big for z.

In summary, it is relevant to outline here that an upcoming activity has been planned with the purpose to replace the off-line training phase with a fully online version that resorts to random features as in Gijsberts et al. [17] and will eventually make the robot learn the eye-hand coordination completely autonomously.

4.1.5 Vision

The remaining piece of information in this journey through the structure of the iCub controller is certainly vision. We strive to provide reliable estimates of object in space since this enables the control of action as presented earlier. One appealing visual cue is motion and we have been recently able to devise a method which provides motion segmentation independent from the movement of the cameras.

Our method is based on the analysis of failures of the standard Lucas-Kanade algorithm [21]. As a general rule, in order to verify that the instant velocity v of a point p has been correctly estimated, the patch W around that point in the image I_t is compared to the patch of the same size at $p+v$ in the new image I_{t+1} (where the original point is supposed to have moved). Given a suitable threshold Θ_M, the discrepancy measure

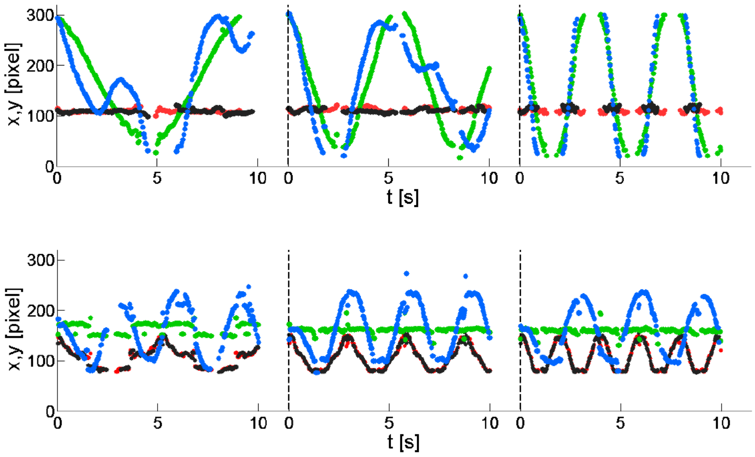


Figure 4.11: Trajectories of the x, y coordinates of the center of mass of the areas detected as moving independently. The cart is moving parallel (up) or orthogonal (down) with respect to the image plane. The plots are reported for the following cart speeds: from left to right 20, 40, $100cm/s$. Colors legend: (1) green for x and red for y in the case of a static head; (2) blue for x and black for y in the case of a head rotating at $20deg/s$.

$$M(p) = \sum_{q \in W} \left(I_t(p+q) - I_{t+1}(p+v+q) \right)^2 , \tag{4.13}$$

is then used to evaluate whether tracking was correctly performed ($M(p) < \Theta_M$) or not ($M(p) \geq \Theta_M$). It is thus interesting to analyze empirically when the Lucas-Kanade algorithm tends to fail and why. Conclusions from this investigation will lead directly to a method to perform independent motion detection. The main empirical circumstances in which errors in the evaluation process of the optical flow arise are three:

- Speed. The instantaneous velocity of the point is too large with respect to the window where motion is being considered. Hence, the computation of temporal derivatives is difficult;
- Rotations. The motion around the point has a strong rotational component and thus, even locally, the assumption regarding the similarity of velocities fails;
- Occlusions. The point is occluded by another entity and obviously it is impossible to track it in the subsequent frame.

Tracking failures caused by high punctual speed depend exclusively on the scale of the neighborhood where optical flow is computed. This issue is usually solved by the so called pyramidal approach which applies the Lucas-Kanade method at multiple image scales. This allows evaluating iteratively larger velocities first and then smaller ones. Instead we determined empirically that when rotations cause failures in the tracking process, this is often a consequence of a movement independent from that of the observer. The third situation in which Lucas-Kanade fails, is caused by occlusions. In this context the main role in determining whether optical flow has been successfully computed is played by the speed at which such occlusion takes place.

We therefore look for points where tracking is likely to fail as soon as one of the conditions discussed is met, i.e. flow inconsistencies due to rotations or occlusions. In detail, we run Lucas-Kanade over a uniform grid on the image, perform the comparison indicated in equation 4.13 and then filter for false positives (isolated failures). The results is a set of independent moving blobs.

We tested the method both in controlled situations (a small robotic device moving linearly in front of the iCub) and, more generally, in tracking people and other moving objects in the laboratory. Figure 4.11 shows results of tracking with both stationary and moving cameras (therefore without and with ego-motion respectively). In the configuration considered, a linear speed of $10cm/s$ corresponds to one pixel

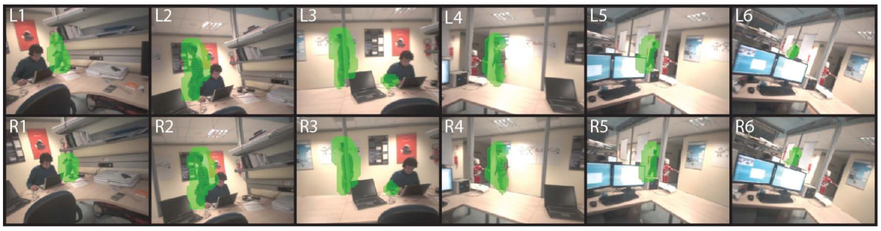


Figure 4.12: A sequence of images recorded during the real time stereo tracking of a person walking in front of the iCub: six images are shown in temporal order from L1 to L6 (left camera) and R1 to R6 (right camera). The walking person is highlighted with a green blob using the result of proposed algorithm.

per frame in a 30 frames-per-second (fps) acquisition. Experiments were conducted up to $100 cm/s$ and with the iCub head adding movement up to $40 deg/s$.

The sequence of images in figure 4.12 is an example of a more naturalistic tracking. In spite of the complexity of the background, it is evident from the images that our method produces robust detection of the moving target with a behavior that varies smoothly in time and is consistent with respect to the two different views acquired from the left and right cameras of the robot. In particular, the movement of the target is effectively tracked both when the person is far from the robot (frames 1 and 6) as well as when he gets closer to it (frames 2-5). Furthermore a substantial modification to light conditions exists with a maximum of brightness reached approximately at frame 4. The algorithm is robust to occlusions: this is visible at the frames in which pillars and posters cover the person. Notably, at frame 3 another person sitting at the table produces a secondary blob with his hand. This distractor is of limited size and it does not interfere with the task since the tracker is instructed to follow the largest blob in the sequence.

These are the data that at the moment the iCub uses for attention, for tracking and which are eventually passed to the reaching controller described earlier. We favored robustness to accuracy here in order to be able to run learning methods and exploration of the environment for considerable periods of time (e.g. as for collecting the 20000 samples mentioned in section 4.1.4.1). Our experiments show that this goal has been fully achieved.

4.1.6 Conclusions

This section of D2.1.1 deals with the problem of building a reliable architecture to control reaching in a humanoid robot – more specifically the iCub – where many degrees of freedom need to be coordinated. We have shown original solutions to vision (using motion), to kinematics (using robust optimization and a multi-referential trajectory formulation) and dynamics (by enabling impedance control from a set of FTSs). Although certain aspects of these methods are somewhat traditional, their specific application and combination is novel. We took particular care in testing all methods rigorously and comparing them with other methods in the literature.

Furthermore, the entire implementation of this software is available, following the iCub/Xperience policies, as open source (GPL) from the iCub SVN repository. These libraries and modules, besides running on the iCub, are available to the research community at large. The algorithms are almost always embedded in static libraries ready to be picked up by others.

The iCub repository can be found at `http://www.icub.org` and browsed on SourceForge (`http://www.sourceforge.net`). This libraries are further linked from the Xperience SVN SourceForge repository. Several videos of the iCub showing the methods described in this deliverable are available on the Internet and in particular at this site: `http://www.youtube.com/watch?feature=player_profilepage&v=LMGSok5tN4A`.

4.2 Edge and surface based grasping affordances

Getting physical control over an unknown object is an important first step to autonomously acquire object properties (e.g., object shape) [19]. In previous work [28] we have therefore shown how visual contour information — extracted using the Early Cognitive Vision System [29], [24] — can can be used to grasp unknown objects with a parallel gripper. In [PKJ$^+$ed] we extended this work to be able to create grasping affordances not only based on contours, but also surface information. We also extended the approach to be able to provide grasps for a three finger hand in addition to the parallel gripper. Both, contour and surface information, can be extracted from a pair of stereo images using the Early Cognitive Vision (ECV) system (see [24] and figure 2.3).

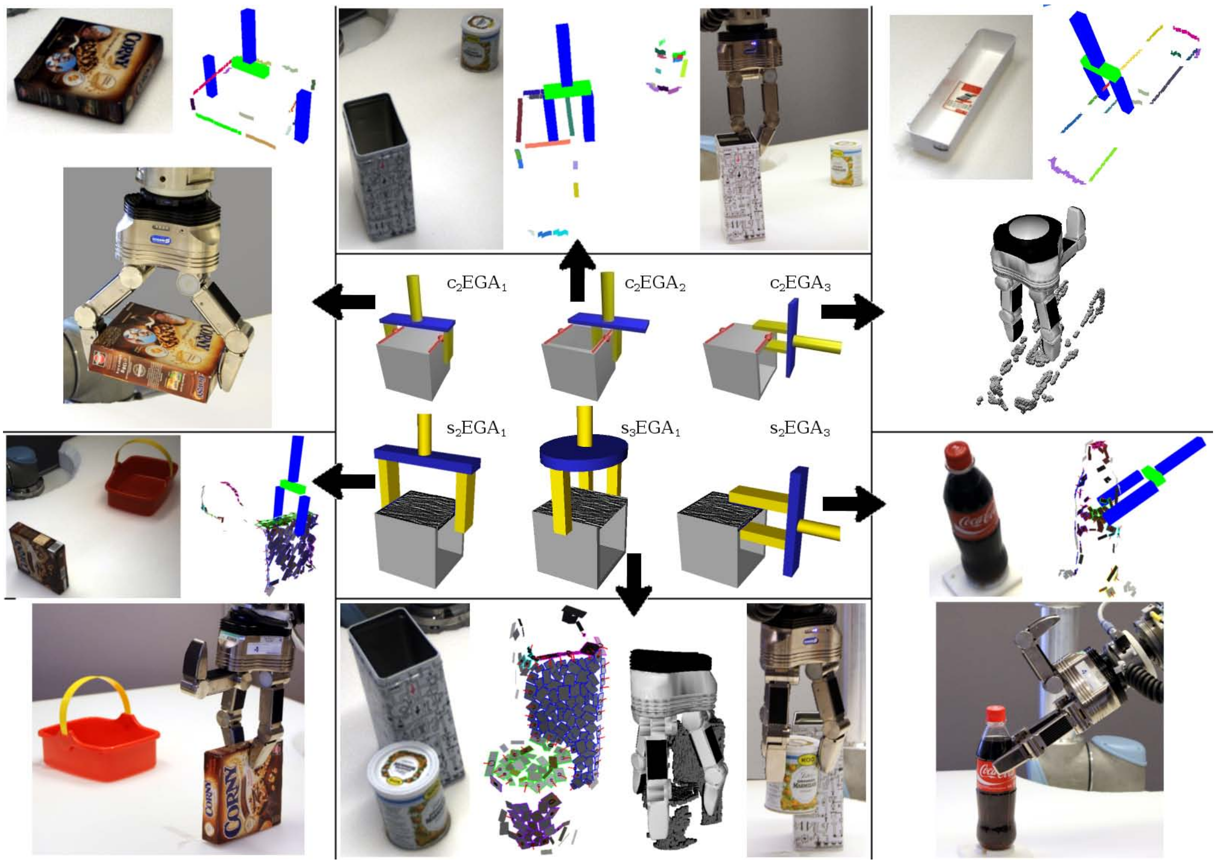


Figure 4.13: Some examples of the elementary grasping actions (EGA). For each type of grasping action, an example is shown consisting of an original image, a snapshot of the ECV representation along with the selected grasp, and the grasp execution in the simulation/real setup.

Based on the surface and contour information extracted using the ECV system elementary grasping actions (EGAs) can be defined. The different sources of structural information allow for different EGAs. The center element of figure 4.13 shows the different EGAs used in this work. The EGAs in the top row are based on two matched contours (highlighted in red) while the bottom row shows grasps based on a surface (darker side of the cube). Figure 4.13 also gives for each EGA a sample scene, the extracted visual representation with the concrete suggested EGA and the execution result. Details about the exact definition of the EGAs and how they can be computed from the visual representation can be found in [PKJ$^+$ed].

The approach was tested in different experimental setups. First, a hybrid real-world and simulated setup was used. Based on real stereo images, our method built a visual representation of the scene and generated grasps. These grasps were then executed in a dynamic simulator. This setup allowed us to test a large number of grasps and get quantitative results, while still dealing with the noise and uncertainty in the real-world visual data. It allowed us, moreover, to compare the different methods under the exact same circumstances. Second, to show the applicability of the proposed methods on real robotic systems, we tested the proposed methods on two different real-world experimental setups, using a parallel and a three-finger dexterous hand.

The results show a good average performance of the proposed grasping methods, which is even further improved when the different methods are combined. This shows that the different methods, which are based on different types of visual information and apply different grasps to contact points, complement each other so that a better performance can be achieved. In particular, the contour-based methods perform better in the situations where objects are not textured, while different surface-based methods complement each other in the situations where objects are textured. By combining both types of information, the overall system can deal with both types of objects. A detailed presentation and discussion of the results can be found in [PKJ$^+$ed].

To enable the community to evaluate and compare different methods for grasping unknown object based on visual information, the above described hybrid real-world and simulated setup is provided as a basis for a benchmark. [KPJ$^+$ed] gives more detail about the concrete setup and how to use the benchmark. The focus of the benchmark here is on the visual processing (extraction of relevant features from the stereo images) and the grasp computation and selection. The user does not have to deal with the complexity on the robotics side (e.g., computation of grasp quality, control of the finger joints in a dynamic setup) since this functionality is provided by the benchmark. The user only needs to provide either (1) wrist position and orientation and a specific hand pre-shape (e.g., 2-finger parallel grasp, 3-finger ball grasp), (2) the finger contact points (the system will compute the hand configuration here based on inverse kinematics) or (3) the wrist position and orientation and the joint angles of all finger joints. The system is then able to evaluate this grasp. The results of the methods described above (from [PKJ$^+$ed]) are given as a baseline for comparisons.

Chapter 5

Conclusions

The project has considerably progressed in defining the new ways how sensorimotor experiences can be achieved by a robot as well as collecting statistics on those sensorimotor experiences. This provides background for more complicated (as compared to reaching and grasping) motor action definition, learning and sequencing and motor-behavior development that is planned for the next period in WP2.1. Cooperation between partners has not been strong while working on this deliverable, as it was not required for this preliminary step of simple action definition and low-level perceptual representation formation. However, cooperation will be required in the next period where there will be a need to put the developed methods together for more complicated action definition.

References

[1] The orocos website. `http://www.orocos.org/kdl`.

[2] The icub website. `http://www.iCub.org`, 2010.

[3] OpenNI framework. `http://www.openni.org`, 2011.

[4] W. Abend, E. Bizzi, and P. Morasso. Human arm trajectory formation. *Brain : a journal of neurology*, 105(Pt 2):331–348, June 1982.

[5] Alexey Abramov, Eren Erdal Aksoy, Johannes Dörr, Florentin Wörgötter, Karl Pauwels, and Babette Dellen. 3d semantic representation of actions from efficient stereo-image-sequence segmentation on gpus. In *International Symposium 3D Data Processing, Visualization and Transmission*, 2010.

[6] E. E. Aksoy, A. Abramov, J. Dörr, K. Ning, B. Dellen, and F. Wörgötter. Learning the semantics of object-action relations by observation. *The International Journal of Robotics Research*, 30(10):1229–1249, 2011.

[7] E. E. Aksoy, A. Abramov, F. Wörgötter, and B. Dellen. Categorizing object-action relations from semantic scene graphs. In *IEEE International Conference on Robotics and Automation, ICRA2010 Alaska, USA*, 2010.

[8] A. Bierbaum, M. Rambow, T. Asfour, and R. Dillmann. A Potential Field Approach to Dexterous Tactile Exploration. In *IEEE/RAS International Conference on Humanoid Robots*, pages 360 – 366, Daejeon, Korea, 2008.

[9] A. Bierbaum, K. Welke, D. Burger, T. Asfour, and R. Dillmann. Haptic Exploration for 3D Shape Reconstruction using Five-Finger Hands. In *IEEE/RAS International Conference on Humanoid Robots*, pages 616 – 621, Pittsburgh PA, USA, Nov 29 - Dec 01 2007.

[10] Alexander Bierbaum, Tamim Asfour, and Rüdiger Dillmann. Dynamic Potential Fields for Dexterous Tactile Exploration. In Rüdiger Dillmann Helge Ritter, Gerhard Sagerer and Martin Buss, editors, *Human Centered Robot Systems*, volume 6 of *Cognitive Systems Monographs*, pages 23–31. Springer Berlin Heidelberg, 2009. HCRS Workshop, Bielefeld.

[11] Alexander Bierbaum, Matthias Rambow, Tamim Asfour, and Rüdiger Dillmann. Grasp Affordances from Multi-Fingered Tactile Exploration using Dynamic Potential Fields. In *IEEE/RAS International Conference on Humanoid Robots*, pages 168 – 174, Paris, France, 2009.

[12] Alexander Bierbaum, Julian Schill, Tamim Asfour, and Rüdiger Dillmann. Force Position Control for a Pneumatic Anthropomorphic Hand. In *IEEE/RAS International Conference on Humanoid Robots*, pages 21 – 27, Paris, France, 2009.

[13] A S Deo and I D Walker. Robot subtask performance with singularity robustness using optimal damped least-squares. In *Proceedings*, IEEE International Conference on Robotics and Automation series, pages 434–441. IEEE, 1992.

[14] Roy Featherstone and David E. Orin. Dynamics. In *Springer Handbook of Robotics*, pages 35–65. Springer, 2008.

[15] Paul Fitzpatrick, Giorgio Metta, and Lorenzo Natale. Towards long-lived robot genes. *Robot. Auton. Syst.*, 56:29–45, January 2008.

[16] T. Gautama and M. Van Hulle. A phase-based approach to the estimation of the optical flow field using spatial filtering. *IEEE Transactions on Neural Networks*, pages 1127–1136, 2002.

[17] Arjan Gijsberts and Giorgio Metta. Incremental learning of robot dynamics using random features. In *ICRA*, pages 951–956, 2011.

[18] M. Hersch and A. Billard. Reaching with Multi-Referential Dynamical Systems. *Autonomous Robots*, 25(1-2):71–83, 2008. The orignal publication is available at springerlink.com.

[19] D. Kraft, N. Pugeault, E. Başeski, M. Popović, D. Kragic, S. Kalkan, F. Wörgötter, and N. Krüger. Birth of the Object: Detection of Objectness and Extraction of Object Shape through Object Action Complexes. *Special Issue on "Cognitive Humanoid Robots" of the International Journal of Humanoid Robotics*, 5:247–265, 2009.

[20] H Y Lee, B J Yi, and Y choi. A realistic joint limit algorithm for kinematically redundant manipulators. In *IEEE International Conference on Control, Automation and Systems*. IEEE, 2010.

[21] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision (ijcai). In *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81)*, pages 674–679, April 1981.

[22] G Metta, D Vernon, and G Sandini. Approach to the development of cognition: Implications of emergent systems for a common research agenda in epigenetic robotics. In 5^{th} *Epigenetic Robotics workshop, Nara, Japan, July 2005*, 2005.

[23] Giorgio Metta, Lorenzo Natale, Francesco Nori, Giulio Sandini, David Vernon, Luciano Fadiga, Claes von Hofsten, Kerstin Rosander, Manuel Lopes, Jos Santos-Victor, Alexandre Bernardino, and Luis Montesano. The icub humanoid robot: An open-systems platform for research in cognitive development. *Neural Networks*, pages 1125–1134, 2010.

[24] Wail Mustafa, Mila Popović, Jeppe Barsøe Jessen, Dirk Kraft, Søren Maagaard Olesen, Anders Glent Buch, and Norbert Krüger. Using surfaces and surface relations in an early cognitive vision system. *(to be submitted)*, 2012.

[25] Duy Nguyen-tuong, Matthias Seeger, and Jan Peters. Computed torque control with nonparametric regression models. In *Proceedings of the 2008 American Control Conference (ACC*, pages 212–217, 2008.

[26] Ugo Pattacini, Francesco Nori, Lorenzo Natale, Giorgio Metta, and Giulio Sandini. An experimental evaluation of a novel minimum-jerk cartesian controller for humanoid robots. In *IROS*, pages 1668–1674. IEEE, 2010.

[27] Karl Pauwels, Norbert Krüger, Markus Lappe, Florentin Wörgötter, and Marc M. Van Hulle. A cortical architecture on parallel hardware for motion processing in real time. *Journal of Vision*, 10(10), 2010.

[28] Mila Popović, Dirk Kraft, Leon Bodenhagen, Emre Başeski, Nicolas Pugeault, Danica Kragic, Tamim Asfour, and Norbert Krüger. A strategy for grasping unknown objects based on co-planarity and colour information. *Robotics and Autonomous Systems*, 58(5):551 – 565, 2010.

[29] N. Pugeault, F. Wörgötter, and N. Krüger. Visual primitives: Local, condensed, and semantically rich visual descriptors and their applications in robotics. *International Journal of Humanoid Robotics (Special Issue on Cognitive Humanoid Vision)*, 7(3):379–405, 2010.

[30] Ali Rahimi and Ben Recht. Random features for large-scale kernel machines. In *In Neural Infomration Processing Systems 20*, 2007.

[31] Lorenzo Sciavicco and Bruno Siciliano. *Modelling and Control of Robot Manipulators (Advanced Textbooks in Control and Signal Processing)*. Advanced textbooks in control and signal processing. Springer, 2nd edition, January 2005.

[32] Bruno Siciliano and Oussama Khatib, editors. *Springer Handbook of Robotics*. Springer, 2008.

[33] Miquel F. Sumsi. *Theory and Algorithms on the Median Graph. Application to Graph-based Classification and Clustering*. PhD thesis, Universitat Autonoma de Barcelona, 2008.

[34] Alvin R Tilley. *The measure of man & woman: human factors in design.* Wiley Interscience, 2002.

[35] N.G. Tsakarakis, G. Metta, G. Sandini, D. Vernon, R. Beira, F. Becchi, L. Righetti, J. Santos-Victor, A.J. Ijspeert, M.C. Carrozza, and D.G. Caldwell. iCub - The Design and Realization of an Open Humanoid Platform for Cognitive and Neuroscience Research. *Journal of Advanced Robotics, Special Issue on Robotic platforms for Research in Neuroscience*, 21(10):1151–1175, 2007.

[36] Andreas Wchter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106:25–57, 2006.

Attached Articles

[KPJ+ed] G. Kootstra, M. Popović, J.A. Jørgensen, D. Kragic, H.G. Petersen, and N. Krüger. Visgrab: A benchmark for vision-based grasping. *Paladyn Journal of Behavioral Robotics*, submitted.

[PAAW12] J. Papon, A. Abramov, E. Aksoy, and F. Wörgötter. A modular system architecture for online parallel vision pipelines. In *Applications of Computer Vision (WACV), 2012 IEEE Workshop on*, 2012.

[PKJ+ed] Mila Popović, Gert Kootstra, Jimmy Alison Jørgensen, Kamil Kuklinski, Konstantsin Miatliuk, Danica Kragic, and Norbert Krüger. Enabling grasping of unknown objects through a synergistic use of edge and surface information. *The International Journal of Robotics Research*, submitted.

[SLP+12] J. Schill, J. Laaksonen, M. Przybylski, V. Kyrki, T. Asfour, and R. Dillmann. Learning Continuous Grasp Stability for a Humanoid Robot Hand Based on Tactile Sensing. In *IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, 2012. (submitted).

[SXP11] Sandor Szedmak, Hanchen Xiong, and Justus Piater. Learning shapes as directed closed surfaces. Technical report, IIS, University of Innsbruck, 2011. It is a draft version of paper to be submitted.

[TP11] Damien Teney and Justus Piater. Probabilistic Object Models for Pose Estimation in 2D Images. In *DAGM*, volume 6835/2011 of *LNCS*, pages 336–345, Heidelberg, 2011. Springer.

VisGraB: A Benchmark for Vision-Based Grasping

Gert Kootstra[1]*, Mila Popović[2],Jimmy Alison Jørgensen[3],Danica Kragic[1],Henrik Gordon Petersen[3],Norbert Krüger[2]

1 Computer Vision and Active Perception Lab, CSC, Royal Institute of Technology (KTH), Stockholm, Sweden

2 Cognitive Vision Lab, The Mærsk Mc-Kinney Møller Institute, University of Southern Denmark, Campusvej 55, DK-5230 Odense, Denmark

3 Robotics Lab, The Mærsk Mc-Kinney Møller Institute, University of Southern Denmark, Campusvej 55, DK-5230 Odense, Denmark

Abstract

We present a database and a software tool, VisGraB, for benchmarking of methods for vision-based grasping of unknown objects. The benchmark is a combined real-world and simulated experimental setup. Stereo images of real scenes containing several objects in different configurations are included in the database. The user needs to provide a method for grasp generation based on the real visual input. The grasps are then planned, executed, and evaluated by the provided grasp simulator where several grasp-quality measures are used for evaluation. This setup has the advantage that a large number of grasps can be executed and evaluated while dealing with dynamics and the noise and uncertainty present in the real world images. VisGraB enables a fair comparison among different grasping methods. The user furthermore does not need to deal with robot hardware, focusing on the vision methods instead. As a baseline, benchmark results of our grasp strategy are included.

Keywords

grasping of unknown objects · vision-based grasping · benchmark

1. Introduction

Grasping previously unseen objects based on visual input is a challenging problem. Various methods have been proposed for solving the problem, as will be discussed later, but it is difficult to compare them and evaluate their strengths and weaknesses. This is due to the fact that methods are often tested on different data and with different hardware setups, which makes it difficult, if not impossible, to repeat the experiments under the same conditions. It is furthermore difficult to quantify results thoroughly, because of the time consuming nature of the experiments. For these reasons, we propose a mixed real-world and simulated benchmark framework. A database of stereo images is provided and the generated grasps are evaluated using a simulated environment, [6, 8], see Figure 1. This setup allows for extensive experimental evaluation, supporting comparison of different methods, while considering noise and uncertainty in the real stereo images. Our previous work used a part of the database as a proof of concept, [14]. In this paper, we present a large database along with software tools to evaluate the generated grasps.

The proposed benchmark focuses on grasping unknown objects in realistic, everyday environments. To deal with noisy and incomplete

*E-mail: kootstra@kth.se

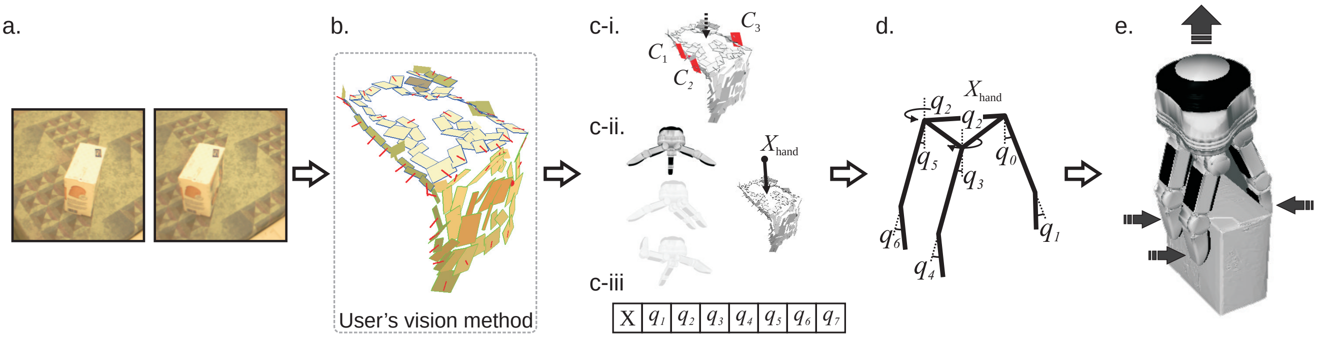


Figure 1. The benchmark pipeline. The real stereo images (a) are input to the user's grasp-generation method (b). Our method is given as a baseline example. The grasp-generation method proposes a grasp hypothesis, either as (c-i) a set of desired contacts , $C = \{C_1, C_2, C_3\}$ (implicitely coding the approach side), (c-ii) by choosing one of the hand pre-grasps and the desired hand pose, or (c-iii) by directly setting the joint angles and hand poser. Based on the grasp hypothesis, the hand pose, $\mathrm{X}_{\mathtt{hand}}$, and configuration, $q = \{q_0, \ldots, q_6\}$, are determined by the provided software (d), and the grasp is executed by the dynamic simulator (e). Note that b) shows the object representation specific to our baseline method (see Section 4).

data coming from robotic sensors and to provide a reduced set of potential grasps, shape primitives, such as boxes and quadrics, have been used in [4, 7]. A less restricted strategy for grasping unknown objects in the real world based on a hierarchical edge representation of the scene has been presented in [15]. In [14], this method has been extended to include surface information. Other methods learn the relation between some visual features and the grasp quality, and apply this knowledge in grasping unknown objects. In [13], for instance, an SVM has been trained to predict the grasp quality based on the hand configuration and the parameters of a single-superquadric representation of the objects. In [5], human expertise is used to learn, the graspability of object parts based on the parameters of superquadrics fitted to segmented parts of the object. Similar, grasp knowledge is learned on a set of simple geometrical shapes and applied to grasp novel objects in [3]. In [18], synthesized objects are used to generate local 2D images features, including edge, texture and color information at different scales, which are used to train the grasp system. The learned knowledge is then used to grasp novel objects in the real world. Features of edge points have been used in [1] to learn to predict grasping success. In [2], the shape context is used for the classification of grasping points.

The presented database contains original stereo images, where no object hypotheses are generated before hand. This means that the grasp generation methods provided by the users of the benchmark need to be able not only to deal with the grasp-generation process but also with generating object hypotheses, if the grasp generation method requires that. Methods such as the one presented in [18] as well as our method [14], works directly on images without the need to generate object hypotheses. There are also several examples that perform figure-ground segmentation at first. In [17], a bottom-up segmentation method based on color and depth information is used to segment the object from its background. Additional information of the table plane is used in [2] to improve object segmentation. In [15], two grasp points are associated with the same surface of an object by using coplanarity and cocolority. Other methods do not need a segmentation of the scene, because they use single image points for pinch grasps, e.g., [18].

Although the studies discussed all deal with grasping unknown objects, the experiments were all done in different conditions; different objects and scenes have been used, as well as different robotic platforms. Some of the studies have been done entirely in simulation (e.g., [3, 5]), whereas others are performed in the real world (e.g., [7, 15, 18]). In this paper, we propose VisGraB as a standardized benchmark for grasping unknown objects based on real-world visual data. By enabling the comparison between different grasping methods, we aim to provide a better insight in the different methodologies and their outcomes.

The wish for a standardized test for grasping has also been put forward in [19], where a benchmark is presented for the evaluation of grasp planners. However, different from our aims, the benchmark focusses on grasping known objects based on full and detailed geometrical information about the objects. We, on the other hand, propose a benchmark for grasping unknown objects in complex scenes based on real, incomplete, and noisy visual observations.

In summary, the main contribution of this paper is the benchmark for vision-based grasping of unknown objects, so that different grasp generation methods can be systematically tested and compared. In addition, the users can focus on the vision aspects, without having to deal with the robotic hardware.

The paper is organized as follows: We first describe the benchmark with the database, the dynamic simulator, and the grasp quality measures in Section 2. In Section 3, a description of how to use the benchmark is given. Next, in Section 4, we give a baseline performance for the benchmark using our method described in [14]. The paper ends with a discussion in Section 5.

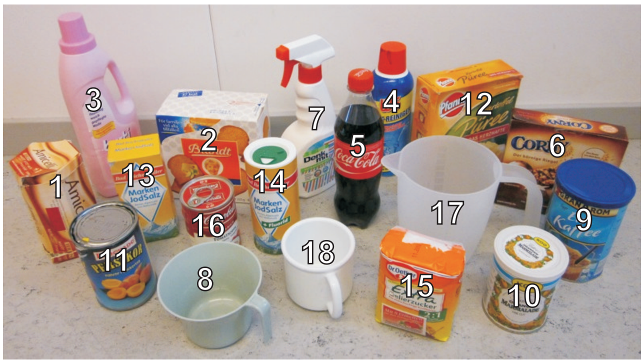


Figure 2. The 18 objects used in the benchmark.

2. The Benchmark

The benchmark consists of a database containing real visual input and a grasp simulator including a dynamics engine to evaluate the grasps. The benchmark contains a total of 432 scenes with a variety of different objects and with different backgrounds. The database includes real stereo images of all the scenes, as well as the 3D models of the scenes, which will be used by the simulator to evaluate the grasps.

The general pipeline of the benchmark is illustrated in Figure 1. Based on the stereo images (Fig. 1a), the user's method generates grasping hypotheses (Fig. 1b). The hypotheses can be provided in different formats (Fig. 1c), as will be discussed in Section 2.3. Given a grasping hypothesis, the software provided with the benchmark determines the pose of the hand and the joint configuration (Fig. 1d). The grasp is then executed by the simulator and the quality of the grasp is displayed to the user (Fig. 1e). Details on the database are given in Section 2.1. Section 2.2 describes the grasp simulator, and the possible grasp representation are given in Section 2.3. Finally, Section 2.4 describes the evaluation of the grasps.

The benchmark, including stereo images, the modeled 3D scenes, and the simulation software can be found on the VisGraB website [9].

2.1. The Database

The 18 objects used in the database are displayed in Figure 2. The objects are part of the KIT ObjectModels Web Database[1]. 3D models of all objects are available for the grasp simulation. The objects have various shapes, sizes, colors, and textures. We recorded scenes with one object and with two objects. In the single-object case, we recorded the 18 different objects in eight different poses, four where the object stands upright, and four where the object lies down. In the double-object scenes, we have 9 combinations of objects, where the objects are in eight different configurations, four where the objects are placed apart, and four where the objects touch each other. All scenes are recorded in two conditions, placed on a non-textured and on a cluttered/textured table. This gives in total $2 \times (18 \times 8 + 9 \times 8) = 432$ scenes. Some example scenes are given in Figure 3, top row.

The scenes are modeled in 3D, in order to test the user-generated grasps in simulation. The models are obtained by calculating the 3D point cloud of the scene using the dense stereo algorithm provided in OpenCV, and subsequently registering the 3D object models to the point cloud using rigid point-set registration [12]. Where necessary, the registration was corrected by hand. A few scene models ae shown in Figure 3, bottom row.

With the database, the vision-based grasping methods are tested for the ability to generate grasps on objects with a variety of different shapes, sizes, colors and textures. Furthermore, the robustness to the pose of the object, the complexity of the scene and the clutter in the scene is tested.

2.2. The Grasping Simulator

The grasps are performed in simulation using RobWork[2], see Figure 1. RobWork is a simulator for robotic grasping with dynamics capabilities, which has been used in several related experiments [8]. The simulator has been initially evaluated in [6], where several thousands of grasps with a parallel gripper in a real robotic setup have been compared to the simulation. Our benchmark methodology can readily be used with other grasp simulators, such as GraspIt [11] and OpenGRASP [10]. The choice of RobWork was mainly motivated by its extensive simulation features such as available hand models, sensors and controllers, but also because of the availability of general robotics tools such as inverse kinematics for hands and robots, trajectory and planning tools.

Using the RobWork grasp simulator including a dynamics engine allows us to not only look at static quality measures of the grasp, but also to determine the actual grasp success by observing the dynamical and physical consequences of the grasp. In our definition, a stable grasp is a grasp with which the object can be lifted without slipping from the hand. We therefore propose a method where the object is lifted after it has been grasped. We therefore define the lift-quality measure (see Section 2.4.1) as an important measure for the stability of a grasp.

We chose to use the three-finger Schunk Dexterous Hand (SDH) for

[1] http://wwwiaim.ira.uka.de/ObjectModels

[2] http://www.robwork.dk

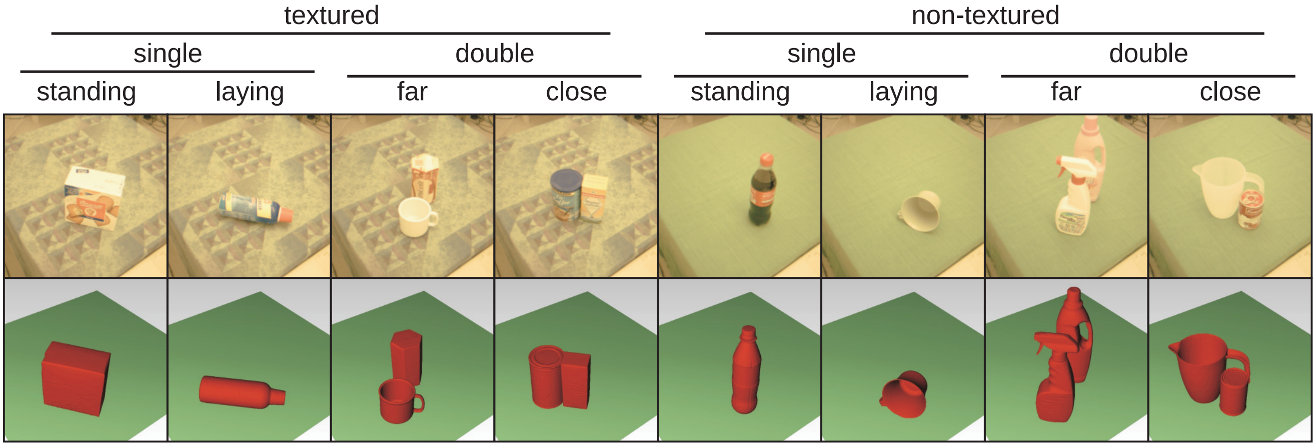


Figure 3. Examples of scenes included in the database. The top row gives the rectified left camera images and the bottom row gives a view on the modeled scenes used for grasp simulation. Examples of the different conditions are given.

the benchmark (see Figure 4), which can be used for both two-finger parallel and three-finger grasping. The SDH has seven degrees of freedom, allowing for complex and flexible grasping. We denote the joint configuration as $\mathrm{q} = \{q_0, \ldots, q_6\}$. Although we made the decision to use the SDH, RobWork supports the easy use of other grippers.

A grasp is performed by first placing the hand in a suitable grasp configuration generated by the user's vision algorithm and using the utility functions provided by the benchmark, see Section 2.3.2 and 2.3.1. The simulation is then started and a grasp-control policy guides the fingers from the start configuration $\mathrm{q}_{\mathrm{open}}$ towards the closed configuration $\mathrm{q}_{\mathrm{closed}}$. When the fingers achieve a static configuration, it is either because of contact forces or because $\mathrm{q}_{\mathrm{closed}}$ is reached. Next, the system attempts to lift the grasped object. After lifting, the quality of the grasp is determined as explained in Section 2.4.1.

The grasp control policy is fairly simple, but can directly be used on the interface of the real hardware of the SDH as well. The policy does not rely on specific sensor feedback other than the joint angles. It requires two joint configurations of the hand $\mathrm{q}_{\mathrm{open}}$ and $\mathrm{q}_{\mathrm{closed}}$, as well as the maximum allowed joint torques τ_{max}. The user moreover needs to provide $\mathrm{X}_{\mathrm{hand}}$, which is the 6-dimensional Cartesian pose of the hand base (position and orientation in 3D). The control policy will close the fingers from $\mathrm{q}_{\mathrm{open}}$ toward $\mathrm{q}_{\mathrm{closed}}$ using a PD controller on each joint. The torque used by the PD controller will be limited by τ_{max} which allows for a rough balancing of the contact forces. As such the simulation only need a few parameters to execute a grasp:

$$(\mathrm{X}_{\mathrm{hand}}, \mathrm{q}_{\mathrm{open}}, \mathrm{q}_{\mathrm{closed}}, \tau_{\mathrm{max}}) \tag{1}$$

These parameters make out the grasp configuration and should be the output of the grasping strategy that is being benchmarked. However, many vision-based grasp strategies do not include grasp control specifics such as inverse kinematics or explicit modeling of joint force limits. To accommodate the need for varying levels of grasp control, the benchmark provides two utility functions that ease the generation of grasp configurations, which are outlined in the next section.

2.3. Grasp Utility Functions

To simplify the generation of grasps, we provide three grasp utility functions as part of the benchmark: based on grasp contacts (Fig. 1c-i), based on hand pre-shapes (Fig. 1c-ii), and based on the the joint configuration (Fig. 1c-iii).

2.3.1. Grasp contacts

The grasp parameters can also be generated by providing two or three desired grasp contacts. See Figure 1c-i for an example of three contacts. A contact $\mathrm{C}_i = \{\mathrm{c}_{\mathrm{pos}}, \mathrm{c}_{\mathrm{dir}}\}$ indicates the position, $\mathrm{c}_{\mathrm{pos}} = \{c_x, c_y, c_z\}$, where the tip of the finger should be placed and the contact direction, $\mathrm{c}_{\mathrm{dir}} = \{c_{d1}, c_{d2}, c_{d3}\}$, which determines in which direction the contact force should work. The inverse kinematics are solved by the utility function provided in the benchmark:

$$\mathrm{C} \mapsto (\mathrm{X}_{\mathrm{hand}}, \mathrm{q}_{\mathrm{open}}, \mathrm{q}_{\mathrm{closed}}, \tau_{\mathrm{max}}) \tag{2}$$

where $\mathrm{C} = \{\mathrm{C}_1, \mathrm{C}_2\}$ for two-finger grasps and $\mathrm{C} = \{\mathrm{C}_1, \mathrm{C}_2, \mathrm{C}_3\}$ for three-finger grasps.

The inverse kinematics algorithm does not require the grasp contacts to be in a specific order or even to be part of the inverse kinematics solution. In the latter case, the algorithm generates

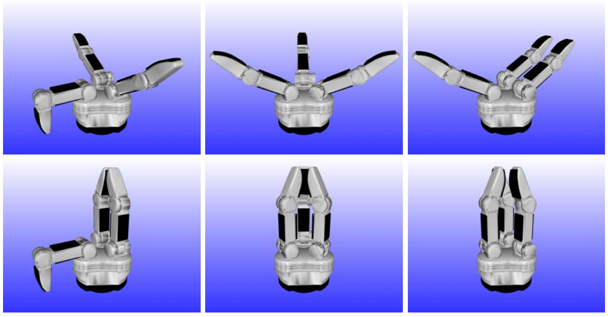


Figure 4. The hand pre-shapes. The top row depicts the $\mathrm{q}_{\mathrm{open}}$ configurations and the bottom row the $\mathrm{q}_{\mathrm{closed}}$ configurations. From the left to the right: 2-finger parallel grasp, 3-finger ball grasp and 3-finger cylinder grasp.

inverse-kinematics solutions that are close to the desired configuration. However, configurations with too high deviation from the target configuration are reported as failed grasps.

2.3.2. Hand pre-shape

It is common to use hand pre-shapes in grasp planning, where the pre-shapes are either generated using simple heuristics or by expert users. For the SDH we have chosen three general hand pre-shapes, see Figure 4. The figure shows the opening and closing positions. The 2-finger parallel grip is shown in left left column, the 3-finger ball grip in the middle column, and the 3-finger cylinder grip in the last column. Given the desired pose of the hand base, $\mathrm{X}_{\mathrm{hand}}$ and the identifier for the specific hand pre-shape, k, the utility function calculates the grasp parameters:

$$(\mathrm{X}_{\mathrm{hand}}, k) \mapsto (\mathrm{X}_{\mathrm{hand}}, \mathrm{q}_{\mathrm{open}}, \mathrm{q}_{\mathrm{closed}}, \tau_{\mathrm{max}}) \tag{3}$$

The complete description of the pre-shape configurations including τ_{max} is available on the VisGraB website [9].

2.3.3. Joint configuration

The user can also use his or her own inverse-kinematic solver to acquire the hand pose, $\mathrm{X}_{\mathrm{hand}}$, and joint configuration when the fingers are in contact with the object, q. The simulation parameters are then obtained with the utility function:

$$(\mathrm{X}_{\mathrm{hand}}, \mathrm{q}) \mapsto (\mathrm{X}_{\mathrm{hand}}, \mathrm{q}_{\mathrm{open}}, \mathrm{q}_{\mathrm{closed}}, \tau_{\mathrm{max}}) \tag{4}$$

2.4. Experimental Evaluation

To test the quality of the user's grasp-generation method, we apply the following experimental procedure: The user provides a list of grasp configuration for every scene in the database. All grasps are then performed by the simulator and the results are returned.

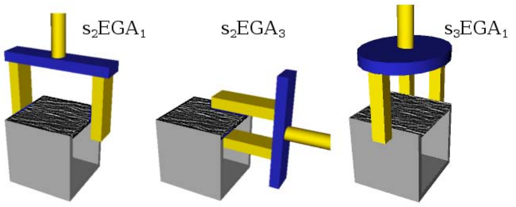


Figure 5. Surface-based Elementary Grasping Actions used by the benchmark method [14].

In a single experimental trial, the quality of the generated grasp is tested as follows: the hand is placed in the correct pose, $\mathrm{X}_{\mathrm{hand}}$. It then closes from the opening configuration, $\mathrm{q}_{\mathrm{open}}$, to the closing configuration, q_{closed}. The object is grasped when the hand settles in a stable configuration and the fingers touch the object. However, this does not necessary mean that the grasp is stable. To test the stability of the grasp, the hand attempts to lift the object. We discriminate the following results:

Stable grasp: The object was grasped and held after lifting, with little or no slippage of the object in the hand.

Object slipped: The object was grasped and held after lifting, but there was considerable slippage of the object in the hand.

Object dropped: The object was grasped, but after lifting, the object was no longer held by the hand.

Object missed: The object was not grasped by the hand.

In collision: The initial hand configuration produced a situation where the hand was penetrating the object(s) and/or the table.

Invalid grasp contacts: The inverse-kinematics solver could not find a joint configuration to reach the desired grasp contacts.

Simulation failure: The simulation failed due to physics-engine failure.

We consider the grasp to be successful when the result is either object slipped or stable grasp. In both cases, the object is in the hand after lifting. The two situations are discriminated based on the amount that the object slipped in the hand during lifting. The slippage defines the lift-quality measure, In case of the double-object scenes, the results are given for the object that is closest to the hand.

2.4.1. Grasp quality measures

In case the object is lifted successfully, we calculate the grasp quality using two quality measures: the lift quality, Q_{lift}, and the grasp wrench-space quality, Q_{gws}.

The lift quality is a dynamic quality measure that represents the ability of a grasp to hold the object stable during lifting, that is, with the object slipping from the hand as little as possible. The lift quality is a value between 0.0 and 1.0 and it is inversely proportional to how much the object moves with respect to the hand during lifting:

$$Q_{\text{lift}} = 1 - \frac{||\mathrm{h} - \mathrm{o}||}{||\mathrm{h}||} \tag{5}$$

where h is the 3D displacement of the hand during lifting and o is the 3D displacement of the object during lifting.
The grasp wrench-space measure Q_{gws} is a static quality measure based upon the grasp wrench space (GWS). The GWS is determined by the friction cones of the contact points, where the friction cone of contact point i is approximated by a set of m contact boundary wrenches, $\{w_{i,j} | j = \{1 \ldots m\}\}$. The six-dimensional contact boundary wrenches are defined as in [11] such that the torque is scaled by the radius r of the object:

$$w_{i,j} = \begin{pmatrix} f_{i,j} \\ \frac{1}{r} \cdot d_i \times f_{i,j} \end{pmatrix} \tag{6}$$

where $f_{i,j}$ is one of the force vectors in contact i, and d_i is the vector from the torque origin to the ith point of contact. The cross product $d_i \times f_{i,j}$ is the torque $\tau_{i,j}$. The GWS is then computed as the convex hull over the union of each set of contact boundary wrenches. Finally, the grasp quality measure Q_{gws} is determined by the radius of the inscribing n-sphere of the GWS, which reflects the maximum perturbrating wrench that the grasp can counterbalance, given the maximum forces of the fingers.

2.4.2. Presentation of results

Scripts are provided as part of the benchmark to parse the simulation results files and to present the results. Since different grasping methods may have their own strengths and weaknesses, we do not summarize the results in a single value. Instead, we give the distribution of grasping results for the different conditions. Furthermore, the grasping quality for each condition is given as the average lift quality, Q_{lift}, and the average grasp wrench-space quality, Q_{gws}, over the successful grasps, i.e., stable grasps and object slipped. An example of the overview of results will be given in Section 4.

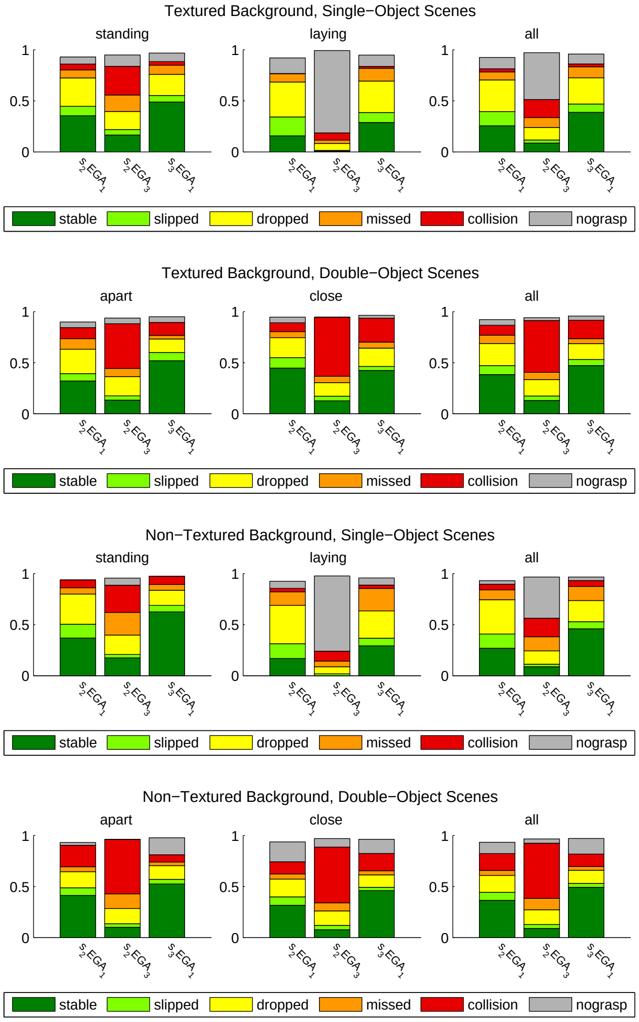


Figure 6. Grasp results. The stacked-bar plots show the average distribution of all grasps over all scenes. The stable and slipped grasps are considered successful grasps, where the object is held in the hand after lifting. The gray area shows the proportion of scenes where the methods do not suggest any grasps.

3. Using the Benchmark

In order to use the benchmark, the database and the software tool need to be downloaded from the VisGraB website [9]. Using the benchmark works in a number of steps:

1. Loading the stereo images and the stereo-calibration file.
2. Generating grasps based on the visual information and providing the grasp configurations, potentially by using the utility functions for hand pre-shapes or grasp contacts.
3. Running the simulation, providing a list of grasp configurations for every scene.
4. Running the scripts to process and represent the results.

The final benchmark results can then be published on the VisGraB website for comparison. The detailed information about the formats and the use of the software can be found on the website.

Table 1. The lift quality and grasp wrench-space quality for the textured scenes. The values are the averages over the successful trials (stable, slipped).

Textured background

		s_2EGA_1		s_2EGA_3		s_3EGA_1	
		Q_l	Q_{GWS}	Q_l	Q_{GWS}	Q_l	Q_{GWS}
Single	standing	0.58	0.46	0.44	0.30	0.72	0.59
	laying	0.31	0.31	0.04	0.03	0.55	0.50
	all	0.44	0.39	0.24	0.17	0.64	0.55
Double	apart	0.60	0.47	0.46	0.35	0.76	0.63
	close	0.68	0.46	0.45	0.35	0.74	0.60
	all	0.64	0.47	0.45	0.35	0.75	0.62

Non-textured background

		s_2EGA_1		s_2EGA_3		s_3EGA_1	
		Q_l	Q_{GWS}	Q_l	Q_{GWS}	Q_l	Q_{GWS}
Single	standing	0.60	0.46	0.51	0.37	0.79	0.65
	laying	0.36	0.31	0.01	0.03	0.55	0.51
	all	0.48	0.39	0.26	0.20	0.67	0.58
Double	apart	0.62	0.50	0.46	0.40	0.68	0.55
	close	0.51	0.41	0.36	0.35	0.68	0.52
	all	0.56	0.46	0.41	0.38	0.68	0.53

4. Baseline Method

To set a baseline for comparison, we used our grasp-generation method presented in [14] and applied it to the VisGraB benchmark. The grasping method is based on an Early Cognitive Vision system [16] that builds a sparse hierarchical representation based on edge and texture information. This representation is used to generate edge-based and surface-based grasps. The method detects surfaces of the objects in the scene, and generates grasps based on these surfaces. For the baseline, we use the surface-based grasps only. The grasp method finds contact points at the boundary of a surface, on which so-called Elementary Grasp Actions are applied, see Figure 5. Based on two grasp contacts, a two-finger encompassing grasp, $s_2\mathrm{EGA}_1$, is generated, as well as two two-finger pinch grasps, $s_2\mathrm{EGA}_3$ one for each contact. Based on three grasp contacts, a three-finger encompassing grasp, $s_3\mathrm{EGA}_1$, is generated. For details about the method, we refer to [14].

4.1. Results

The grasp results are shown in Figure 6 and the grasp quality of the successful grasps are in Table 1. The results indicate that the three-finger encompassing-grasps are most successful, followed by the two-finger encompassing-grasps. Due to missing visual information about the back of the objects, the two-finger pinch grasps results more often in collisions or no grasp is suggested. In general, the methods are more successful in grasping one object from the double-object scene then grasping the object in the single-object scene. However in the double-object scenes there are more collisions. The results for the scenes with textured and non-textured background are veru similar, which shows that out method can deal with a higher degree of visual complexity. The grasp success for the individual objects are given in Tables 2 and 3.

5. Discussion

We presented VisGraB, a database and a software tool for benchmarking vision-based grasping of unknown objects. The database contains real stereo images, which can be used by the user to generate grasp hypotheses. These hypotheses can then be passed on to the software tool, which contains a dynamic grasps simulator that plans, executes, and tests the grasp. The database contains a large set of scenes, with different objects displaying a variety of different shapes, sizes, colors and textures, and with different backgrounds. By performing the grasps in simulation, a large number of grasps can be repeatedly tested. The benchmark facilitates 1) the evaluation and comparison of different vision-based grasp-generation methods in a standardized fashion, and 2) a focus on the vision methods instead of on the robotic hardware. We presented an example as an illustration of the use of the benchmark.

In addition to what we presented here, the VisGraB framework can be used for evaluating a variety of tasks related to grasping, for example grasping known objects can be tested using the KIT object models, and learning methods can be evaluated on their generalization abilities.

We strongly encourage the use of the benchmark to test your vision based grasp-generation methods and to compare it to other methods. We are very open to extend the benchmark based on future needs from the community.

Acknowledgement

This work was supported by the EU through the projects eSMCs (FP7-IST-270212), XPERIENCE (FP7-ICT-270273), and CogX (FP7-ICT-215181), by the Swedish Foundation for Strategic Research through the project RoSy.

References

[1] L. Bodenhagen, D. Kraft, M. Popović, E. Baśeski, P. E. Hotz, and N. Krüger. Learning to grasp unknown objects based on 3d edge information. In Proceedings of the 8th IEEE international conference on Computational intelligence in robotics and automation, 2009.

[2] J. Bohg and D. Kragic. Learning grasping points with shape context. Robotics and Autonomous Systems, 58(4):362–377, 2010.

[3] N. Curtis and J. Xiao. Efficient and effective grasping of novel objects through learning and adapting a knowledge base. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008.

[4] C. Dune, E. Marchand, C. Collowet, and C. Leroux. Active rough shape estimation of unknown objects. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008.

[5] S. El-Khoury and A. Sahbani. Handling objects by their handles. In Proceedings of IROS 2008 Workshop on Grasp and Task Learning by Imitation, 2008.

[6] L. Ellekilde and J. Jorgensen. Usage and verification of grasp simulation for industrial automation. In Proceedings of 42st International Symposium on Robotics (ISR), Chicago, 2011.

[7] K. Hübner, S. Ruthotto, and D. Kragic. Minimum volume bounding box decomposition for shape approximation in robot grasping. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'08), pages 1628–1633, 2008.

[8] J. Jorgensen, L. Ellekilde, and H. Petersen. Robworksim - an open simulator for sensor based grasping. In Proceedings of Joint 41st International Symposium on Robotics (ISR 2010) and the 6th German Conference on Robotics, Munich, 2010.

[9] G. Kootstra, M. Popović, J. A. Jørgensen, D. Kragic, H. G. Petersen, and N. Krüger. VisGraB: A benchmark for vision-based grasping. http://csc.kth.se/visgrab.

[10] B. León, S. Ulbrich, R. Diankov, G. Puche, M. Przybylski, A. Morales, T. Asfour, S. Moisio, J. Bohg, J. Kuffner, and R. Dillmann. Opengrasp: a toolkit for robot grasping simulation. In Proceedings of the Second international conference on Simulation, modeling, and programming for autonomous robots, SIMPAR'10, pages 109–120, Berlin, Heidelberg, 2010. Springer-Verlag.

[11] A. T. Miller and A. T. Miller. Graspit!: A versatile simulator for robotic grasping. IEEE Robotics and Automation Magazine, 11:110–122, 2004.

[12] C. Papazov and D. Burschka. Stochastic optimization for rigid point set registration. In Proceedings of the 5th International Symposium on Visual Computing (ISVC'09), volume 5875 of Lecture Notes in Computer Science, pages 1043–1054. Springer, 2009.

[13] R. Pelossof, A. Miller, P. Allen, and T. Jebara. An svm learning approach to robotic grasping. In Proceedings of the IEEE Conference on Robotics and Automation, 2004.

[14] M. Popović, G. Kootstra, J. A. Jørgensen, D. Kragic, and N. Krüger. Grasping unknown objects using an early cognitive vision system for general scene understanding. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2011.

[15] M. Popović, D. Kraft, L. Bodenhagen, E. Başeski, N. Pugeault, D. Kragic, T. Asfour, and N. Krüger. A strategy for grasping unknown objects based on co-planarity and colour information. Robotics and Autonomous Systems, 58(5):551 – 565, 2010.

[16] N. Pugeault, F. Wörgötter, and N. Krüger. Visual primitives: Local, condensed, and semantically rich visual descriptors and their applications in robotics. International Journal of Humanoid Robotics (Special Issue on Cognitive Humanoid Vision), 7(3):379–405, 2010.

[17] D. Rao, Q. V. Le, T. Phoka, M. Quigley, A. Sudsang, and A. Y. Ng. Grasping novel objects with depth segmentation. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2010.

[18] A. Saxena, J. Driemeyer, and A. Y. Ng. Robotic grasping of novel objects using vision. The International Journal of Robotics Research, 27(2):157–173, 2008.

[19] S. Ulbrich, D. Kappler, T. Asfour, N. Vahrenkamp, A. Bierbaum, M. Przybylski, and R. Dillmann. The opengrasp benchmark suite: An environment for the comparative analysis of grasping and dexterous manipulation. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2011.

Textured background

Single-object scenes

	s_2EGA_1		s_2EGA_3		s_3EGA_1	
	s	l	s	l	s	l
1	50 %	10 %	12 %	8 %	76 %	31 %
2	28 %	74 %	4 %	0 %	77 %	86 %
3	17 %	32 %	21 %	6 %	9 %	19 %
4	48 %	15 %	38 %	0 %	37 %	21 %
5	57 %	38 %	50 %	0 %	33 %	12 %
6	45 %	44 %	9 %	0 %	67 %	12 %
7	23 %	76 %	43 %	0 %	35 %	64 %
8	33 %	44 %	0 %	0 %	31 %	13 %
9	23 %	9 %	16 %	0 %	38 %	55 %
10	87 %	29 %	31 %	4 %	97 %	61 %
11	60 %	47 %	20 %	0 %	59 %	73 %
12	54 %	60 %	5 %	0 %	76 %	47 %
13	54 %	38 %	22 %	0 %	59 %	58 %
14	74 %	4 %	46 %	0 %	86 %	35 %
15	14 %	19 %	12 %	0 %	63 %	41 %
16	90 %	50 %	64 %	0 %	82 %	65 %
17	33 %	25 %	0 %	0 %	13 %	0 %
18	13 %	0 %	0 %	8 %	53 %	0 %

Double-object scenes

	s_2EGA_1		s_2EGA_3		s_3EGA_1	
	f	c	f	c	f	c
a	58 %	87 %	0 %	0 %	84 %	85 %
b	21 %	48 %	0 %	0 %	40 %	55 %
c	0 %	32 %	36 %	33 %	45 %	31 %
d	37 %	48 %	24 %	23 %	42 %	24 %
e	52 %	66 %	43 %	30 %	76 %	61 %
f	44 %	54 %	20 %	25 %	43 %	39 %
g	31 %	33 %	15 %	27 %	68 %	36 %
h	53 %	76 %	8 %	9 %	76 %	55 %
i	58 %	50 %	10 %	8 %	67 %	31 %

Table 2. Percentage of successful grasps for the different objects in the textured scenes. Results for the single-object scenes are split into standing (s) and laying (l) object poses and for the double-object scenes into far (f) and close (c). The pairs in the double-object scenes are: a: 1-18, b: 2-11, c: 3-7, d: 4-15, e: 5-14, f: 6-8, g: 9-13, h: 10-12, i: 16-17.

Non-textured background

Single-object scenes

	s_2EGA_1		s_2EGA_3		s_3EGA_1	
	s	l	s	l	s	l
1	72 %	25 %	7 %	0 %	93 %	30 %
2	37 %	69 %	4 %	0 %	85 %	88 %
3	34 %	51 %	23 %	0 %	49 %	10 %
4	69 %	5 %	28 %	0 %	53 %	12 %
5	52 %	21 %	31 %	0 %	48 %	13 %
6	43 %	19 %	8 %	0 %	66 %	6 %
7	41 %	63 %	23 %	0 %	46 %	61 %
8	25 %	50 %	13 %	0 %	21 %	0 %
9	48 %	33 %	10 %	0 %	85 %	61 %
10	86 %	19 %	17 %	0 %	99 %	72 %
11	70 %	46 %	20 %	0 %	89 %	74 %
12	33 %	54 %	6 %	0 %	74 %	47 %
13	70 %	29 %	21 %	15 %	91 %	56 %
14	73 %	13 %	24 %	0 %	87 %	39 %
15	45 %	14 %	10 %	0 %	86 %	19 %
16	78 %	44 %	66 %	0 %	91 %	53 %
17	19 %	0 %	16 %	18 %	19 %	0 %
18	13 %	10 %	50 %	0 %	57 %	20 %

Double-object scenes

	s_2EGA_1		s_2EGA_3		s_3EGA_1	
	f	c	f	c	f	c
a	71 %	79 %	3 %	3 %	93 %	82 %
b	42 %	24 %	2 %	0 %	44 %	53 %
c	29 %	4 %	17 %	4 %	16 %	28 %
d	38 %	45 %	18 %	18 %	38 %	54 %
e	66 %	63 %	27 %	21 %	83 %	48 %
f	77 %	55 %	18 %	22 %	38 %	51 %
g	25 %	44 %	12 %	17 %	64 %	80 %
h	57 %	20 %	7 %	3 %	63 %	19 %
i	35 %	26 %	19 %	19 %	74 %	27 %

Table 3. Percentage of successful grasps for the different objects in the non-textured scenes. Results for the single-object scenes are split into standing (s) and laying (l) object poses and for the double-object scenes into far (f) and close (c). The pairs in the double-object scenes are: a: 1-18, b: 2-11, c: 3-7, d: 4-15, e: 5-14, f: 6-8, g: 9-13, h: 10-12, i: 16-17.

A Modular System Architecture for Online Parallel Vision Pipelines

Jeremie Papon, Alexey Abramov, Eren Aksoy, Florentin Wörgötter
Bernstein Center for Computational Neuroscience (BCCN)
III Physikalisches Institut - Biophysik, Georg-August University of Göttingen
{jpapon,abramov,eaksoye,worgott}@physik3.gwdg.de

Abstract

*We present an architecture for real-time, online vision systems which enables development and use of complex vision pipelines integrating any number of algorithms. Individual algorithms are implemented using modular plugins, allowing integration of independently developed algorithms and rapid testing of new vision pipeline configurations. The architecture exploits the parallelization of graphics processing units (GPUs) and multi-core systems to speed processing and achieve real-time performance. Additionally, the use of a global memory management system for frame buffering permits complex algorithmic flow (*e.g.* feedback loops) in online processing setups, while maintaining the benefits of threaded asynchronous operation of separate algorithms. To demonstrate the system, a typical real-time system setup is described which incorporates plugins for video and depth acquisition, GPU-based segmentation and optical flow, semantic graph generation, and online visualization of output. Performance numbers are shown which demonstrate the insignificant overhead cost of the architecture as well as speed-up over strictly CPU and single threaded implementations.*

1. Introduction

There is a growing interest in development of complex vision systems for robotic vision applications. Such research has strict requirements; these systems must operate in real-time, using input from multiple sources, and typically consist of multiple algorithms which work in concert to produce useful output with minimal delay. Consequently, the architecture which binds algorithms and input sources together has become an increasingly important factor. This work presents a vision architecture which uses modular plugins, a novel buffering scheme, and GPU memory optimizations to allow real-time performance of an online vision system, even with complex pipelines and algorithms developed by independent researchers.

A primary concern when developing such complex vision systems lies in how to properly integrate algorithms developed by different researchers, often from multiple institutions. Typically, computer vision researchers develop solutions tailor-made for their particular problem, without concern over the difficulties involved in integrating their particular algorithm in a large system. The proposed architecture eases this integration process by providing a plugin interface. The plugin system allows independently developed algorithms to communicate with the architecture's central memory management system, interact with the GUI, define their own unique data types, and integrate into systems with plugins developed by other researchers.

Another motivation for developing a vision architecture is the desire to enable the use of complex algorithmic layouts in an online system. In particular, interest in creating loops that allow high level algorithms (*i.e.* which come late in the pipeline) to feedback and improve the output of low level vision methods. Traditional online vision pipeline architectures cannot accommodate such loops in an adequate way, as at any given moment each portion of the pipeline is processing data from different instants in time.

Existent vision system architectures also do not support the use of GPUs in a fully integrated way, leading to inefficient use of the device and communication with device memory. The presented method incorporates specially designed GPU data-containers to ensure optimal PCI-bus use through a pre-caching scheme and concurrent memory transfers. In addition to these, extendibility is ensured through an interface which allows user-defined data-container handling, allowing plugin developers to explicitly define how the memory manager shares data between the host and device. The paper is structured as follows: first we review existing architectures, then present our system, describe a typical system configuration used for robotics, and then give performance figures from a demonstration setup.

2. Related Work

There are a few existing open-source projects centered around computer vision system architecture, such as iceWing [**?**] and Imalab [**?**]. These systems bear some similarities to ours, in that they are sophisticated vision development environments, featuring modularity, efficient visualization, and simple control of algorithm parameters. While a step forward, these projects lack two core features required for our work; support of feedback loops and integrated use of the GPU as a coprocessor. In addition to the open-source projects, there are a few commercial solutions available. Foremost among these is MATLAB, which uses a high-level scripting language to allow for rapid development. Unfortunately, its restrictive and expensive licensing can make it difficult to develop algorithms in distributed locations; every developer must have not only a MATLAB license, but also licenses for the multiple toolboxes required. Additionally, since MATLAB (and it's open source equivalent Octave) development is not in C/C++, creation of novel GPU algorithms using a language such as CUDA is difficult. Other commerical solutions, such as HALCON [?] or BLOX [?] also suffer from their restrictive licensing, making them not well suited for research. None of these solutions permit feedback loops in a real-time online vision system.

3. System Architecture

Our vision system is a plugin shell which provides an easy-to-use API for interacting with the GUI, memory management system, and visualization components. In order to ensure expandability, such a system must provide straightforward communication and interaction between plugins created independently, while employing strong-typing checks to ensure only valid plugins may be interconnected. In addition, it must ensure that plugins have the flexibility to define their own methods for visualization. Finally, the system must ensure that each plugin is self-contained, and executes within its own thread. This is especially important for fast execution on modern processors, where the number of cores can match, or even exceed, the number of plugins one is running.

In the next subsections, we shall describe how our architecture accomplishes these goals while requiring as little computational and communication overhead as possible. Small overhead is especially important in the case of real time video processing, where relatively large images must be processed at fast frame rates.

3.1. Execution Flow

At its core, the architecture provides a shell which consists of a GUI for loading plugins and visualizing data, a system for storing plugin output to file, and a buffering/memory-management system for handling data. This functionality is contained in the *Main Thread* and *Memory Manager Thread* shown in Figure 1. Users build their system by adding plugins, configuring their options via the GUI, and then connecting the plugins to each-other. The user can also save/load a fully configured system as an XML file. Once a vision system has been built, the user can control execution using the frame rate module, which controls the firing rate of the system clock.

As the whole system runs asynchronously in independent threads, the clock trigger acts as the initial starting point for each frame. This means that any source plugins, such as a stereo camera rig or a video file reader plugin, must connect to the frame rate module. As a trigger arrives at each plugin, a triggering signal is sent to the memory manager, telling it to generate a *DataContainer* for the plugin's output. The plugin is then triggered, causing it to execute its processing functionality and generate output, which it stores in the location assigned to it by the memory manager. The plugin then generates another triggering signal, which is connected to both the memory manager and whatever ensuing plugins use the output as their input. When a plugin has multiple inputs, it will loop inside its execution thread, waiting until all inputs for a frame have arrived before executing. This is accomplished by each thread having its own input queue map; it is important to note though, that these queues contain no actual data (and thus minimal overhead), and merely serve as a message passing system. The signaling and triggering system employs the open-source Qt signal & slot architecture. In particular, the system makes use of Qt's ability to queue signals for execution as they arrive at a thread.

3.2. Plugin Development and Interaction

The functionality of the system is provided primarily via plugins. A plugin consists of a shared library which is located and loaded dynamically at run-time. The system is based on the low-level Qt plugin API, which facilitates development and ensures compatibility across different platforms. Plugins inherit from a pure abstract interface class which defines a protocol for communicating with the core application. This permits plugins to define input and output types and pass messages to/from the GUI and memory manager.

Developers are required to implement a *processData* function, which receives input and writes to an output *DataContainer*. The developer can optionally create any number of GUI elements (*e.g.* sliders, buttons) using the interface functions. Plugins specify how many inputs they require, and give the possible types for these inputs. Communication between plugins is accomplished through a standardized data container interface. The core architecture contains commonly used data container implementations, such as *StereoImageContainer*. Plugins may define their own specialized data containers which are loaded at runtime with the plugin. For example, the Segmentation plugin has its own container type *SegmentationData*, which contains a list of labeled segments, metadata about the segments, and la-

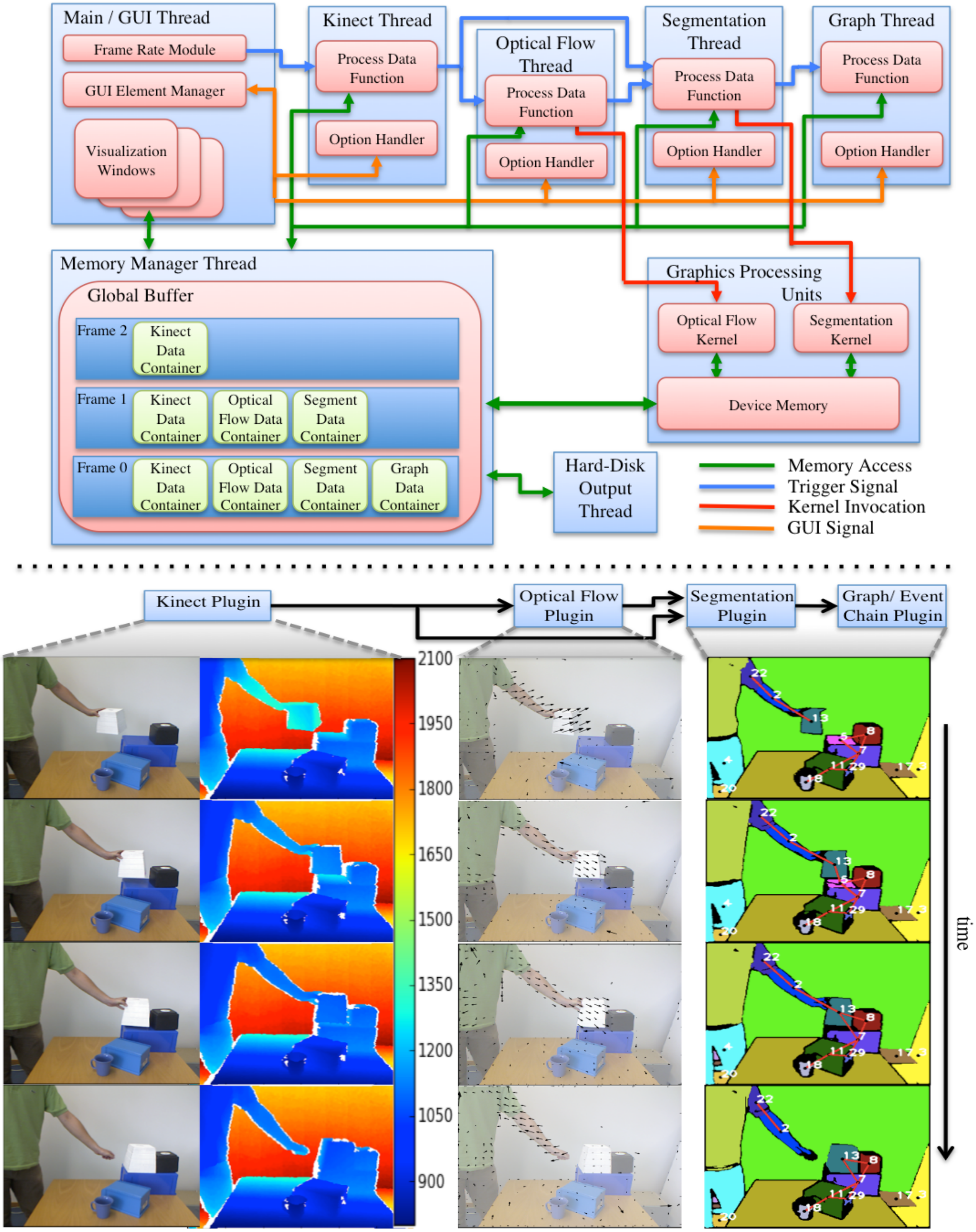


Figure 1. Overview of the system architecture and demonstration system output for four frames. The colums show output from the different components; from left to right, Kinect image and depth (in mm), optical flow, and graphs overlaid on segmentation plugin output. This type of output can be seen live in any number of visualization windows within the GUI.

beled images. The standardized data container interface allows for any plugin to refer to a new container class without actual knowledge of the container itself other then the string identifiers of its members (*e.g.* ”Segment Labels”). Correct handling of access to these members is accomplished through dynamic dispatch using the virtual lookup table. This ensures that a plugin written by one researcher can be easily used as input to another’s, as long as they know the proper identifiers and underlying formatting of the data.

3.3. Visualization

During the development and use of a vision system, it is of utmost importance to be able to visualize what is occurring at every stage of the system pipeline. As such, our system allows users to create any number of visualization windows which can select any plugin to display (and which part of the plugin’s output to display, *e.g.* left or right image). If a developer creates their own data container for a plugin, they can define a special visualization callback function as part of this container. The system will automatically detect this callback when the plugin is loaded, and use it for visualizing the plugin’s output. Developers can specify multiple methods for visualizing the plugin; the GUI for visualization will allow selection of which to display.

Visualization windows read directly from the global buffer, and as such have a small memory overhead. Additionally, visualization runs in the GUI thread, rather than in any of the plugin threads. If a plugin slows down the system, visualization (and the GUI) will remain responsive, allowing the user to troubleshoot. This also means that visualization that requires computation, such as labeling an image with text or vector graphics, will have a negligible effect on the actual frame throughput of the system. If visualization lags behind the system output, frames are automatically skipped on an interval that allows visualization to maintain synchronization with the rest of the system. This is of particular importance in an online system, such as our real-time robotic application, where visualization lagging behind processing can cause confusion or even errors.

4. Memory Architecture

The memory management system has been designed to allow distributed development and computing, complex system pipelines incorporating feedback loops, and efficient use of the GPU as a computational resource. The following subsections will describe how these design goals have been achieved by illustrating our *Global Buffer* design and explaining how it manages GPU memory.

4.1. Global Buffer

Our global buffer concept was designed to overcome the limitations of standard online vision pipelines. In a standard online pipeline a local buffering scheme is used; each algorithm has an input buffer, where data accumulates while it is waiting to be processed. Such a setup is adequate as long as the pipeline remains unidirectional, but complications arise in using feedback loops. Figure 2 compares a standard pipeline with our global buffer; unlike a typical buffering scheme, our global buffer maintains and manages all memory in a central location (and separate thread). The global buffer is responsible for dynamic allocation of all data containers, maintaining reference counts, and determining when a frame can expire. Since the global buffer is responsible for maintaining memory, plugins use a message passing system to communicate. Plugins pass messages to each other to notify completion of a new frame, or to trigger a feedback mechanism. They also use the message passing system to request that the global buffer allocate a new data container for their output. When a developer creates a new type of data container, they use a simple interface to pass the global buffer a function pointer for creating an instance of their new data container type.

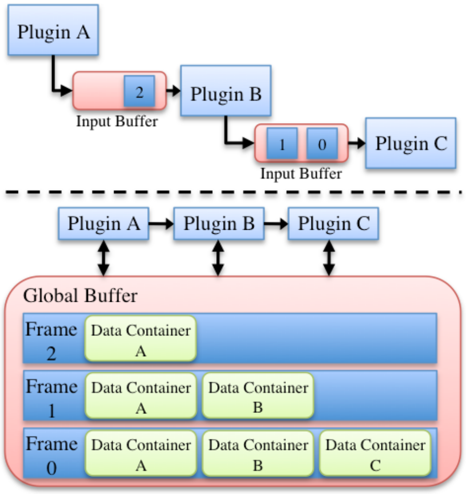


Figure 2. A typical buffering scheme (top) and our buffer (bottom).

In order to fully understand the limitations of a standard buffering system, consider, for instance, the system shown at the top of Figure 3. If the feedback mechanism is triggered for frame n, plugin B must return to frame n in order to modify how it was processed. This is not possible in the standard local buffer scheme, as that data was discarded after it was used as input to B. One possible solution is to maintain another local buffer for each plugin which contains data which has already been processed, but this quickly adds several degrees of complexity. In particular, garbage collection becomes very difficult, and management of these buffers when feedback does occur becomes unnecessarily convoluted.

The global buffer solves this by maintaining data in a

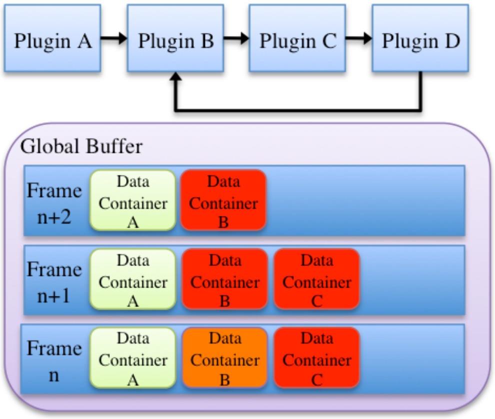


Figure 3. Feedback using a global buffer

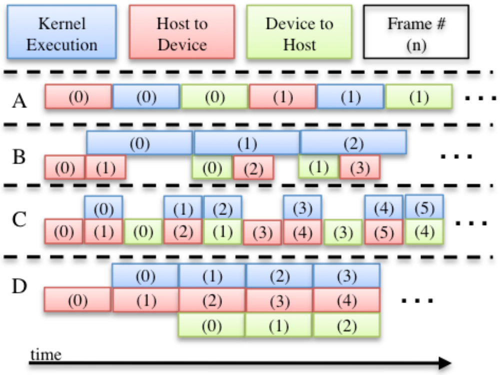


Figure 4. Streaming; Concurrent kernel execution

more structured way. When a feedback mechanism is triggered for frame *n* the triggering plugin (*D*) sends a message to *B*, causing it to stop processing what it has scheduled, and revert to frame *n*. As frame *n* is still easily accessible in the global buffer, *B* can simply send a request for the pointer(s) to the input data container(s) it requires. The global buffer is guaranteed to still have the data for frame *n*, because *D* never produced an output for frame *n*, so the global buffer has not marked frame *n* as complete. Once *B* finishes processing frame *n* with its new feedback information, it will overwrite its old output for frame *n* (shown in orange) and then simply continue on as it would normally, processing frame *n+1*. The feedback corrected data will propagate down the pipeline, and any data which is no longer valid (shown in red) will simply be overwritten. Infinite feedback loops are avoided by a preventing feedback from occurring more than once per plugin per frame.

4.2. GPU Memory Handling

While utilizing the massively-parallel GPU as a coprocessor has become increasingly common, how to integrate it effectively into an open vision architecture remains an open question. Particularly vexing is how to integrate it seamlessly into the memory system of such an architecture, as the GPU has separate physical memory, which is entirely distinct in both location and structure from that used by the CPU [**?**]. Data streaming through the system must be transferred to the GPU for modules which use it, and then transferred back out for visualization and used by modules later in the pipeline.

A naive implementation of this architecture would simply serialize the operations; when a module needs to use the GPU, it copies data to device memory, executes a kernel, and then copies the output back out to host memory. While this is still relatively efficient, it fails to fully take advantage of the pipelined streaming architecture, since the memory transfer bandwidth is idle while the kernel is executing. The architecture uses the streaming CUDA API to utilize this spare bandwidth, allowing it to perform concurrent asynchronous memory transfer and kernel execution.

As shown in Figure 4, we utilize a pre-caching technique, whereby data for frame *n+1* is transferred during the execution of frame *n*. When the kernel execution time is significantly longer than the transfer time (*B*), memory transfer is completely hidden, even with unidirectional memory. When kernel execution time is comparable to memory transfer time, only some of the transfer can be hidden (*C*), unless the hardware supports concurrent data transfers[1] (*D*).

5. Demonstration System

This section presents a real-time demonstration system, consisting of six plugins. The demonstration system calculates dense disparity using a standard stereo camera setup (rather than Kinect data) in order to show the flexibility of the architecture as well as highlight the speedup achieved via multithreading. Switching from Kinect input to a stereo camera setup is simply a matter of changing connections in the GUI. The pipeline described consists of plugins for reading and rectifying stereo data, calculating optical flow[**?**], computing disparity[**?**], segmentation and tracking[**?**], dense disparity estimation, and semantic graph and event chain generation[**?**, **?**]. This type of a system configuration is used to recognize and learn object manipulation actions in a robotics context.

[1] Concurrent data transfers are supported under the Fermi architecture[**?**]. Currently the Fermi Quadro and Tesla series cards have two Direct memory access (DMA) engines[**?**], allowing them to perform host-to-device and device-to-host operations simultaneously. The consumer Fermi cards (GTX 4xx, 5xx) only have a single DMA engine, so concurrent transfers are disabled on them.

5.1. Image Acquisition

Video is acquired using a Firewire stereo camera rig. Triggering for image acquisition can be controlled using either an external hardware trigger or the architecture's software clock. Rectification is performed on the GPU (there is a separate plugin for calibration using a standard chessboard). Time from triggering to output of a rectified pair of stereo images is around 10ms at 1024x768.

5.2. Disparity and Optical Flow

Optical flow is computed using the GPU implementation [**?**] of a phase-based algorithm [**?**]. The algorithm tracks the temporal evolution of equi-phase contours by taking advantage of phase constancy. Differentiation of the equi-phase contours with respect to time yields spatial and temporal phase gradients. Optical flow is then computed by integrating the temporal phase across orientation. Estimates are refined by traversing a Gabor pyramid from coarser to fine levels. The plugin uses the five most recent frames to compute optical flow in the case of online video, but can also use ”future” frames when working with recorded movies (this can slightly improve the quality of output flow).

Sparse disparity maps are computed on the GPU using a technique similar to optical flow [**?**]. Rather than use temporal phase gradients, the disparity algorithm relies on phase differences between stereo-pair rectified images. As with the optical flow algorithm, results are computed using a coarse to fine pyramid scheme.

5.3. Segmentation and Tracking

The segmentation and segment tracking plugin has two roles; first, it partitions the image into labeled regions, as seen in the right-most column of Figure 1, and second, it determines correspondences between frames to maintain consistent labeling. The segmentation algorithm is based on the work of Blatt *et al*. [**?**], which applies the Potts model in such a way that superparamagnetic phase regions of aligned spins correspond to a natural partition of the image data. Initial spins are assigned to pixels randomly, and then a Metropolis-Hastings algorithm with annealing [**?**] is used to iteratively update the spins until an equilibrium state is reached.

The Metropolis algorithm is implemented on the GPU[**?**], permitting real-time performance. The algorithm itself lends itself to efficient implementation on a GPU, as interactions are only computed locally (8 connected nearest-neighbors). Coupling interactions between pixels are determined using average color vector difference (in the HSV space) of nearest-neighbors. Additionally, when depth data is available, the algorithm prevents interactions between pixels if there is a significant difference in their depth values. This prevents coupling across regions which have similar color but discontinuous depth.

In addition to segmentation, the plugin maintains consistent labels for objects from frame to frame. This is accomplished by transferring spins between frames using output from an optical-flow plugin [**?**]. As such, only the first frame is actually initialized at random; subsequent frames are initialized using a forward-propagated version of the previous frame's equilibrium spins. This has two advantages; the number of iterations needed to reach equilibrium is greatly reduced since the spin distribution already approximates the final state, and the algorithm naturally tracks objects since spins (and thus labels) are maintained over time.

5.4. Semantic Graphs

The semantic graphs plugin constructs a symbolic 3D description of the scene from the segmentation results and disparity maps. Segments are used to construct undirected and un-weighted graphs (seen in the right-most column of Figure 1; nodes are labeled with numbers and red lines are graph edges). Each segment is given a node and edges represent their three dimensional touching relations. Graphs can change by continuous distortions (lengthening or shortening of edges) or, more importantly, through discontinuous changes (nodes or edges can appear or disappear). Such a discontinuous change represents a natural breaking point: All graphs before are topologically identical and so are those after the breaking point. Hence, we can apply an exact graph-matching method [**?**] at each breaking point and extract the corresponding topological main graphs. The sequence of these main graphs thus represents all structural changes (manipulation primitives) in the scene.

This type of graph representation is then encoded by a semantic event chain (SEC), which is a sequence-table; rows and columns of which represent possible spatial relations between each segment pair and manipulation primitive. This final output can be used to classify manipulations and categorize manipulated objects for use in a robotics or human-computer interaction (HCI) setting[**?**, **?**]. The primary advantage of this method is that actions can be analyzed without models or a-priori representation; the dynamics of an action can be acquired without needing to know the identities of the objects involved.

6. Results and Discussion

Testing was performed to compare single threaded with multi-threaded operation mode and to detect the impact of visualization on processing speed. Testing was performed on an Intel i7 (3.33Ghz, 8 execution threads) system with an NVIDIA GTX 295 GPU. The demonstration setup depicted at the top of Figure 5 was used for all tests. To determine if visualization had a negative impact, the tests were run with and without a visualization windows for each component, showing live views of their outputs. Timing measurements

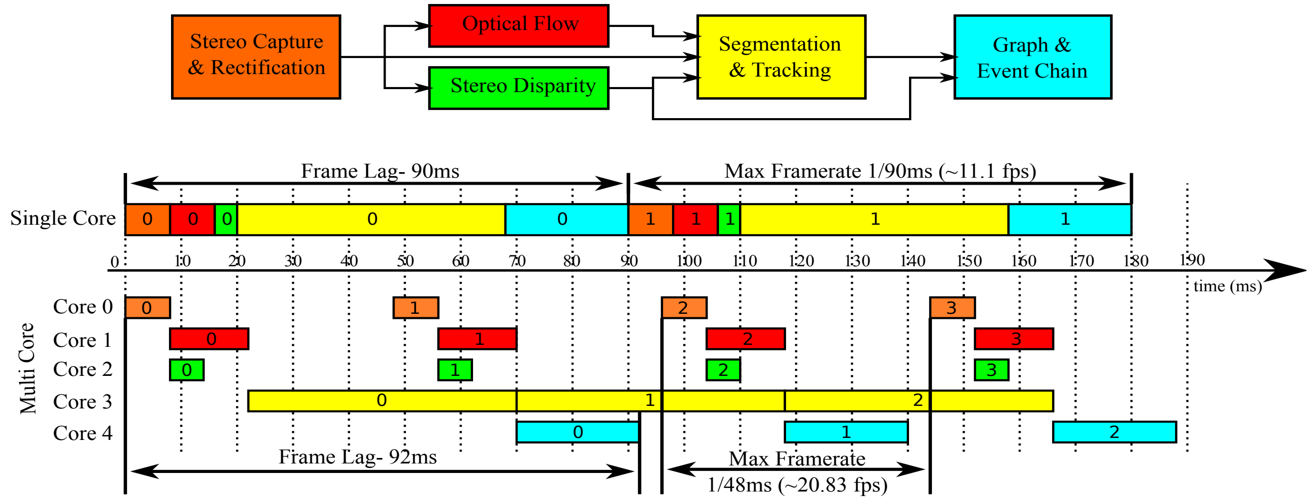


Figure 5. Timing results for demonstration system; plugins are color coded and contain frame numbers. When run in single thread mode, short GPU operations such as optical flow are significantly faster due to reduced overhead; this results in slightly lower (2ms) frame lag. The true benefit of multi-threaded mode is the higher maximum frame-rate that can be achieved.

for plugins are the mean execution time per frame of a 1000 frame (640x480) stereo video sequence (frames of which are shown in Figure 1), averaged over 10 runs. The code for the single and multi-threaded versions is identical with the exception of the movement of plugin objects to separate threads.

We measure performance by analyzing two key attributes of a pipelined vision real-time vision system. First, in terms of frame lag, that is time from frame acquisition to final output, multi-threaded mode is slightly slower than single-threaded. As shown in Figure 5, this is due to relatively fast plugins which use the GPU (disparity and optical flow in this case). This can be attributed to the static overhead cost incurred by switching between threads while using the CUDA run-time API. The switching is relatively expensive for short GPU operations as it forces the CUDA driver to create and destroy GPU contexts[2]. This could be avoided by the addition of an additional GPU; in our demonstration system the driver is forced to change contexts as there are three threads (flow, disparity, segmentation) attempting to use two GPUs. Additionally, the architecture will soon be brought to the newest CUDA release, which allows context sharing between threads. It should also be noted that at higher resolutions multi-threaded mode overtakes single-threaded, as the overhead cost of context switching is outweighed by the gain from computing optical flow and disparity in parallel.

The second measure of performance, throughput, or maximum frame rate, shows a significant speedup in multi-threaded mode, almost doubling from 11.1 (stereo)fps to 20.83. While significant, the speedup is not equal to the number of execution threads used by the demonstration setup (six; one for each plugin and one for the GUI & memory manager). This less-than-optimal gain can be attributed to the fact that the demonstration system had one component, segmentation & tracking, which was significantly slower then the rest. As seen in Figure 5, the entire system throughput is limited by the rate at which the segmentation plugin produces output.

As seen in Figure 6, the addition of visualization components has a small impact on performance. This delay was most noticeable for the shorter components, disparity and optical flow, but never exceeded 2ms. Fortunately, this additional time does not affect throughput in multi-threaded mode, as it is hidden by the length of the longest component. The times with visualization were used for Figure 5; clearly shortening the time of any component other than segmentation will have a negligible effect on performance. While the increase does not affect throughput, it has a slight effect on frame lag. Frame lag is less important than throughput for our research, but it should be noted that in certain cases, such as when quick reactions are required, frame lag may be an important performance measure.

Although we have shown that an architecture which supports feedback loops for an online vision pipeline can be implemented efficiently and can have real-time performance, we have not presented a feedback loop in our demonstration system. The description of the algorithms which use them is beyond the scope of this paper. As such, we presented the buffering system which enables the use of feedback loops in a pipeline, but leave testing of the efficacy of feedback

[2] GPU contexts are analogous to CPU processes, and each have their own distinct address space. Each thread may only have one context active at a time, and contexts may not share threads. See [?, ?] for more details.

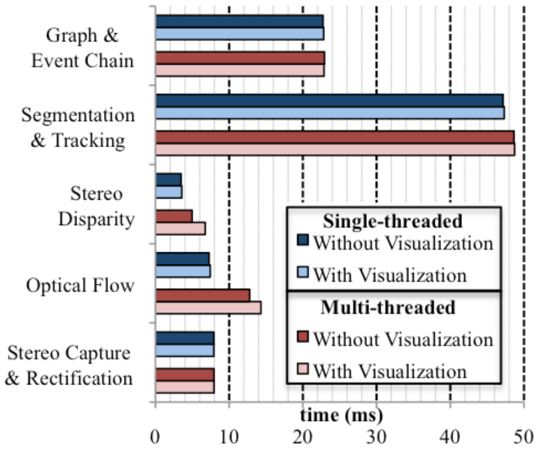


Figure 6. Visualization has a slight impact on performance, but the effect is negligible in multi-threaded mode where the slight increases in processing time are hidden in the length of the longest component (in this case, segmentation).

mechanisms themselves in improving segmentation results to future publication.

7. Conclusion

Building a self-contained, efficient, and complete vision system acts as a significant barrier to entry for those wishing to develop and test new vision algorithms. We have presented a modular plugin environment, designed specifically for expandability and parallel architectures, which facilitates rapid distributed development of vision pipelines. Our plugin system allows simple collaboration between organizations, allowing developers to share algorithms easily, and without forcing them to share code. The architecture permits streaming use of the GPU as a coprocessor, efficient visualization of algorithm outputs, and the ability to use complex pipelines involving feedback mechanisms. The system architecture is being released under an open-source GPL license[3], with the goal of spurring the growth of GPU use and the research of feedback mechanisms in real-time vision applications by lowering the cost-to-entry of development and prototyping of algorithms.

Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 (Specific Programme Cooperation, Theme 3, Information and Communication Technologies) under grant agreement no. 270273, Xperience and grant agreement no. 269959, Intellact.

[3] https://launchpad.net/oculus

References

[1] F. Lomker, S. Wrede, M. Hanheide, and J. Fritsch, "Building modular vision systems with a graphical plugin environment," in *Computer Vision Systems, 2006 ICVS '06. IEEE International Conference on*, p. 2, jan. 2006.

[2] A. Lux, "The imalab method for vision systems," in *International Conference on Vision Systems, ICVS-03*, pp. 21–26, 2003.

[3] "MVTec halcon - building vision for business." http://www.mvtec.com/halcon/, 2011.

[4] "Common vision BLOX imaging system." http://en.commonvisionblox.de/, 2011.

[5] "NVIDIA's next generation cuda compute architecture: Fermi." Whitepaper, http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf, 2009.

[6] NVIDIA, "NVIDIA quadro dual copy engines." Whitepaper, http://www.nvidia.com/docs/IO/40049/Dual_copy_engines.pdf, 2010.

[7] K. Pauwels, N. Krger, M. Lappe, F. Wörgötter, and M. M. Van Hulle, "A cortical architecture on parallel hardware for motion processing in real time," *Journal of Vision*, vol. 10, no. 10, 2010.

[8] A. Abramov, E. Aksoy, J. Dörr, F. Wörgötter, K. Pauwels, and B. Dellen, "3d semantic representation of actions from efficient stereo-image-sequence segmentation on gpus," in *International Symposium 3D Data Processing, Visualization and Transmission*, 2010.

[9] E. E. Aksoy, A. Abramov, F. Wörgötter, and B. Dellen, "Categorizing object-action relations from semantic scene graphs," in *IEEE International Conference on Robotics and Automation, ICRA2010 Alaska, USA*, 2010.

[10] E. E. Aksoy, A. Abramov, J. Dörr, K. Ning, B. Dellen, and F. Wörgötter, "Learning the semantics of object-action relations by observation," *The International Journal of Robotics Research (IJRR), Special Issue on 'Semantic Perception for Robots in Indoor Environments' (In press)*, 2011.

[11] T. Gautama and M. Van Hulle, "A phase-based approach to the estimation of the optical flow field using spatial filtering," *IEEE Transactions on Neural Networks*, pp. 1127–1136, 2002.

[12] M. Blatt, S. Wiseman, and E. Domany, "Superparamagnetic clustering of data," *Physical Review Letters*, vol. 76, no. 18, pp. 3251–3254, 1996.

[13] M. F. Sumsi, *Theory and Algorithms on the Median Graph. Application to Graph-based Classification and Clustering*. PhD thesis, Universitat Autonoma de Barcelona, 2008.

[14] NVIDIA Corporation, "NVIDIA CUDA C programming guide," 2010. Version 3.2.

Enabling grasping of unknown objects through a synergistic use of edge and surface information

Mila Popović[a] Gert Kootstra[b] Jimmy Alison Jørgensen[c] Kamil Kuklinski[d]
Konstantsin Miatliuk[d] Danica Kragic[b] Norbert Krüger[a]

December 20, 2011

[a] the Cognitive Vision Lab, the Mærsk Mc-Kinney Møller Institute, University of Southern Denmark, Odense, Denmark
[b] the Computer Vision and Active Perception Lab, CSC, Royal Institute of Technology (KTH), Stockholm, Sweden
[c] the Robotics Lab, The Mærsk Mc-Kinney Møller Institute, University of Southern Denmark, Odense, Denmark
[d] Automation and Robotics Dept. Białystok University of Technology, Poland

Abstract

Grasping unknown objects based on real-world visual input is a challenging problem. In this paper, we present an Early Cognitive Vision system that builds a hierarchical representation based on edge and texture information, which provides a sparse but powerful description of the scene. Based on this representation, we generate contour-based and surface-based grasps. We test our method in two real-world scenarios, as well as on a vision-based grasping benchmark providing a hybrid scenario using real-world stereo images as input and a simulator for extensive and repetitive evaluation of the grasps. The results show that the method generates a large percentage of successful grasps, and in particular that the edge and surface information are complementary. This allows our approach to deal with rather complex scenes.

Keywords: Vision-based grasping, Grasping unknown objects, Visual scene representation, Dexterous hands

1 Introduction

In this paper we propose a vision system for general scene understanding allowing for grasp planning. We focus on grasping unknown objects for which top-down knowledge is not available. In contrast to 2D approaches, which often need simplifying assumptions on the actual action execution, e.g., (Saxena et al., 2008; Chinellato et al., 2005), we make use of 3D information in terms of contour and surface descriptors, allowing for improved grasp planning. In contrast to other 3D approaches that are based on segmenting scenes by different kinds of shape primitives, e.g., (Hübner et al., 2008; Miller et al., 2003), our approach does not require any kind of complex segmentation and registration process nor manual preprocessing, but operates on elements of a visually extracted hierarchical representation of the scene.

One of the problems in grasp planning is the nearly infinite number of possible grasps, which all need to be evaluated to assess their quality. Many current approaches therefore reduce the number of possible grasps by modeling the object shape with a number of shape primitives, such as boxes (Hübner et al., 2008), cylinders, cones, spheres (Miller et al., 2003), or superquadrics (Goldfeder et al., 2007). With the approach we present here, such explicit shape abstractions are not necessary. Our vision system inherently provides a sparse and abstract, but powerful set of 3D features. Making use of our hierarchical representation of the scene, the amount of computed grasps can be controlled by the granularity of the feature descriptors at the different levels of the hierarchy. Moreover, our 3D features are naturally aligned with the shape of the object, which is not necessary the case when shape primitives are used. This allows our method to plan grasps more accurately.

More specifically, we propose and evaluate a method for the bottom-up generation of two- and three-fingered grasps based on contour and surface structures. These structures are extracted by means of an extension of the biologically-motivated hierarchical vision system (Pugeault et al., 2010a). This system, in the following called Early Cognitive Vision (ECV) system, makes use of an elaborated mid-level ECV stage in which structurally rich and disambiguated information is provided to higher levels of visual processing (for a detailed discussion see (Krüger et al., 2010)). This system has been applied to the problem of grasping unknown objects based on contour relations (Popović et al., 2010). The ECV system has not only been used for grasping, but also for tasks such as pose estimation (Detry et al., 2009; Kjær-Nielsen et al., 2010), and object learning and recognition (Pugeault et al., 2010a; Başeski et al., 2010).

In this paper, we extend the ECV system, which primarily was dealing with edge-like structures (Pugeault et al., 2010a), by texture information to allow for the association of grasps

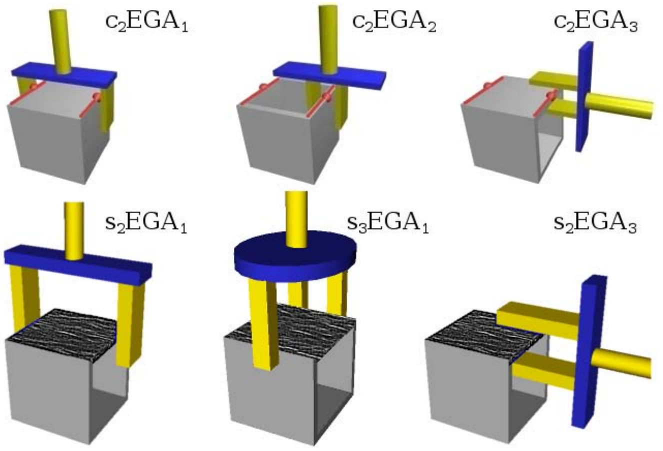


Figure 1: The elementary grasping actions (EGA). Top row: three types of contour-based EGAs. The red lines indicate the selected contours. Bottom row: three types of surface-based EGAs. The dark face shows the selected surface. The first letter in the naming scheme marks the type of features used to generate a grasp. 'c' stands for contour-based and 's' for surface based. The following subscript stands for two or three fingers. The last subscript marks the general type of grasp, where '1' is an encompassing grasp, '2' is a pinch grasp from the top and '3' is a pinch grasp from the side of the surface.

to surface information in addition to the contour information. The different sources of structural information allow for different elementary grasping actions (EGAs). The top row of Fig. 1 shows three EGAs based on two matched contours: two-finger encompassing grasps, top pinch grasps, and side pinch grasps, such as presented in (Popović et al., 2010). The bottom row shows three surface-based EGAs for two and three-finger encompassing grasps, and side pinch grasps. Figure 2 shows the different EGAs together with the examples of the performed experiments.

The ECV system provides a visual hierarchical representation, where the perceptual organization is guided by 2D and 3D geometrical and appearance relations between visual entities at the different levels of the hierarchy, see Fig. 3. In the edge domain, the hierarchy consists of local edge primitives, grouped into contours, which are then matched to contours belonging to the same surface. In the texture domain, local textured patches (texlets) are grouped into larger surface elements (surflings), which are further grouped into surfaces. The hierarchy is illustrates in Fig. 3) and described in more detail in Sect. 3.1. The visual features allow for the extraction of orientation and depth discontinuities, which can be used for surface segmentation. Once the 3D surfaces with their boundaries have been extracted, we generate the EGAs by associating a set of grasping hypotheses to a single boundary primitive or to duplets or triplets of boundary primitives.

We do systematic tests of the generated grasp hypotheses in two scenarios. First by using the vision-based grasping benchmark, VisGraB (Kootstra et al., submitted), which provides a mixed real-world and simulated environment, in which features are extracted from real visual data and grasps are performed in a virtual environment using a dynamic simulation (see Fig. 11). This allows us to test a large number of grasps generated from natural stereo images (in total over 40,000 grasps were tested). By that we can make elaborated quantifications of contour-based and surface-based grasps. The second scenario is a real-world scenario with a stereo camera, an industrial robot arm, a gripper, and real objects. Grasps are tested in two different hardware setups, one with the Schunk parallel jaw gripper and other with the Schunk three-finger dexterous hand, see Fig. 4.

This paper makes the following three contributions: (1) We extend the ECV system with a hierarchy of features in the texture domain, providing a sparse and meaningful representation of the scene. The representation on one side reduces the search space for grasping and on the other side creates additional context information which is relevant for grasping, (2) we define new grasping affordances based on texture and surface information and build a system that performs the grasps, and (3) we show the complementary strength of edge and texture information for grasping in an extensive experimental evaluation.

This article is a major extension of the conference article (Popović et al., 2011). We go beyond (Popović et al., 2011), in a number of ways: by presenting a novel surface-based grasp-generation method, by giving a much more detailed description of applied methods, and finally by performing elaborate experiments in a mixed real-world and simulated setup and on two different real robotic setups.

The paper is organized as follows: Section 2 discusses the related work in the area of vision-based grasping. In Section 3 we introduce the ECV system. Section 4 presents the three grasp generation algorithms and details on the grasp execution procedure are given in Section 5. We describe the experimental evaluation scenarios and give the overview of the results in Section 6. The paper is concluded with a discussion in Section 8.

2 Related Work

Different approaches to visual-based object grasping have been proposed by the robotic community. As proposed in (Bohg and Kragic, 2010), these approaches can be roughly divided into grasping of *known*, *familiar*, and *unknown* objects.

For grasping known objects, a detailed 2D or 3D model of the object is generally available. This model is then fitted to the current visual observation to retrieve the pose of the object. Based on the model and the pose estimation, a large number of grasps suggestions are generated and their quality is evaluated to select the most promising grasp, e.g., (Nguyen, 1989; Shimoga, 1996). One of the main challenges is the huge amount of possible grasps. In order to reduce the search space, different techniques have been applied. In (Miller et al., 2003; Goldfeder et al., 2007; Hübner et al., 2008), the shape of the object is simplified by using shape primitives, such as, spheres, boxes, and superquadrics, thereby reducing the number of possible grasps. Another method for reducing the dimensionality of the configuration space of the hand, using so-called eigengrasps, has been proposed in (Ciocarlie and Allen, 2009). It has been demonstrated in (Borst et al., 2003) that generating an optimal grasp, according to some quality measure, is not

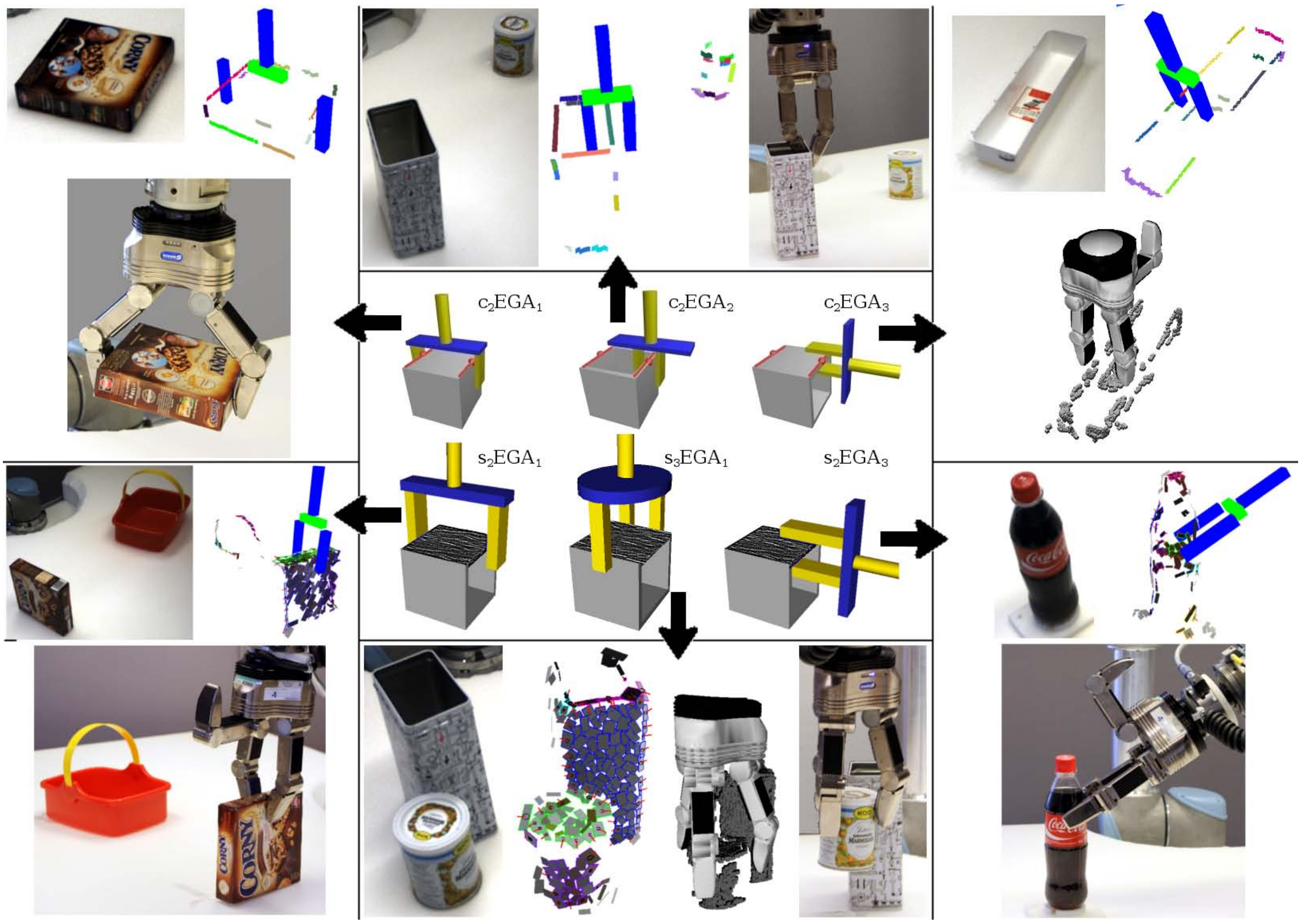


Figure 2: Some examples of the elementary grasping actions (EGA). For each type of grasping action, an example is shown consisting of an original image, a snapshot of the ECV representation along with the selected grasp, and the grasp execution in the simulation/real setup.

necessary. A small random subsample of the possible grasps will generate grasps of the average quality which is sufficient. In (Detry et al., 2011) grasp affordances are modeled with continuous probability density functions (grasp densities) which link object-relative grasp poses to their success probability. Grasp data is gathered from exploration and when a satisfactory number of data is available, an importance-sampling algorithm turns these into a grasp density.

The above-mentioned studies have been done in simulation, assuming complete knowledge about the object and the robot, and ignoring noise, with the exception of (Detry et al., 2011) and (Hübner et al., 2008), where incomplete and noisy data has been used as well. The studies all assume a perfect segmentation of the object from its background. In contrast, we propose a method based on real visual data without any knowledge about the presented objects.

In work on the grasping of familiar objects, the system is generally trained on a set of objects and learns the relation between some visual features and the grasp quality. This knowledge is then used to grasp resembling objects. In (El-Khoury and Sahbani, 2008), for instance, the graspability of object parts is learned based on the parameters of superquadrics fitted to segmented parts of the object and using human expertise. An SVM has been trained to predict the grasp quality based on the hand configuration and the parameters of a single-superquadric representation of the objects in (Pelossof et al., 2004). In (Curtis and Xiao, 2008), grasp knowledge is learned on a set of simple geometrical shapes and applied to grasp novel objects. All these experiments were done completely in simulation with synthesized data. In (Saxena et al., 2008) grasping hypotheses are learned based on a set of local 2D images features using synthesized objects, and this knowledge is used to grasp objects in the real world. The feature vector used is a high dimensional set of edge, texture and color features on different scales. Different features of two contours resulting from our ECV system have been used in (Bodenhagen et al., 2009) to learn to predict the grasping success.

When grasping unknown objects, no model of the objects or prior grasp knowledge is used and all reasoning about grasping is done on the visual observation of the scene. In (Hübner et al., 2008) and (Dune et al., 2008), shape primitives, respectively boxes and quadrics, were used to deal with the noisy and incomplete data coming from robotic sensors, and to provide a reduced set of potential grasps. In (Gallardo and Kyrki, 2011), the stereo-vision data was approximated with box and cylinder primitives, which describe an objects overall shape, as well as

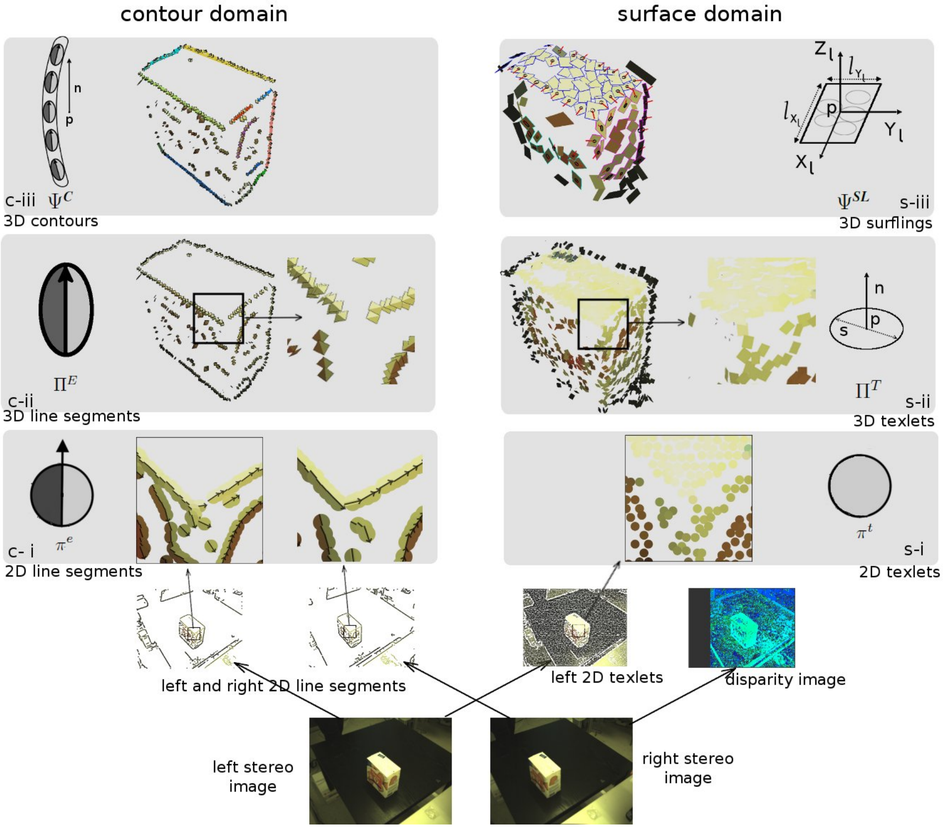


Figure 3: The hierarchical representation of contour and texture information in the ECV system. This figure is best viewed in color.

its location, orientation, and size. Grasps based on the 2D silhouettes of the objects were tested in (Morales, 2004). A learning framework for discovering the visual features that predict a reliable grasp was presented. In (Richtsfeld and Vincze, 2008), a top surface of an object was detected by selecting the points that belong to the top 3 mm of the segmented point cloud. The grasping points were placed at the rim of the detected surface and the grasps were performed in simulation. A sophisticated 3D representation of the scene based on ECV system was used in (Kraft et al., 2008; Popović et al., 2010) for grasp planning. We build upon this work by extending the system to not only take edge features into consideration, but also texture features.

Most of the studies on vision-based grasping assume a segmentation of the objects from their background. However, for grasping unknown objects in real-world situations this assumption does not hold. When using pinch grasps, (as done in e.g., (Saxena et al., 2008)), single image points are sufficient to define a grasp, making object segmentation to relate multiple points on the same object unnecessary. Input of the user is taken to initialize object segmentation using active contours in (Dune et al., 2008). In (Rao et al., 2010), a bottom-up segmentation method based on color and depth information is used and the graspability of the segments is learned using an SVM. In (Popović et al., 2010), we associated two grasp points with the same surface of an object by using co-planarity and co-colority.

In this paper, we present vision-based bottom-up methods for grasping unknown objects, based on unsegmented real-world scenes. Unlike other approached discussed in this section, we do not use a simplification of the object(s) using shape primitives to abstract the shape. Instead we extend the ECV system to produce a sparse, yet semantically meaningful representation of the scene that remains close to the true shapes of the objects and which allows the system to utilize the potential of edge as well as texture information.

3 Theoretical Framework

In this section we describe the Early Cognitive Vision (ECV) framework. We specifically introduce the visual features that are used for grasp generation, i.e. edges, contours, texlets, surflings and surfaces. The edges and contours are described briefly and we refer to our previous work, while the surface extraction is given in a more detail to allow for the precise formalization of the novel surface-based grasp methods.

3.1 The ECV System

The framework of the Early Cognitive Vision system provides a rich hierarchical visual representation that includes edges and textured surfaces (Krüger et al., 2010; Pugeault et al., 2010a). Fig. 3 shows different stages in creating the hierarchical representation. The left-hand branch illustrates the propagation of edge information, and the right-hand branch shows the propagation of texture information.

The representation is layered and starts with extracting sparse local features from 2D images. These basic features are called *multi-modal primitives* and encode both geometric and appearance information. In this paper, we use *edge primitives* and textured-surface primitives (*texlets*). By matching 2D features over two stereo views, the system derives corresponding 3D descriptors for the different structures, see Fig. 3.

On the second level, the ECV system organizes basic features into perceptual groups (in both 2D and 3D) (Başeski et al., 2010; Pugeault et al., 2010b). Edge primitives are grouped into *contours*, while the texlets are organized in so-called *surflings*, see Fig. 3. On this higher level of abstraction it is again possible to group complex features or observe their relations. Contours are matched to other contours that are part of the same surface and surflings are combined into *surfaces*, see Fig. 5b.

3.1.1 Edges and Contours

Line segments in the ECV system are local edge-feature descriptors (see Fig. 3c-i,c-ii) that integrate geometrical (position and orientation) and appearance (color and phase) information, see (Pugeault et al., 2010a). The local edge features are grouped into bigger perceptual groups – contours – based on multi-modal constraints including proximity, collinearity, co-circularity, and similarity in appearance. (see Fig. 3c-iii). Contours, as features on the higher level of abstraction, can again be compared and grouped by observing relations between them. In this paper, as in (Popović et al., 2010), we use the co-planarity and co-colority contour relations to identify pairs of contours belonging to the same surface of an object. These are used to trigger contour-based grasping actions, see Sect. 4.1 and Figs. 1,2(top row). Co-planarity refers to contours that lie in the same plane, while co-colority means that contour share the same color, for details see (Başeski et al., 2010).

3.1.2 Texlets, Surflings and Surfaces

The full formalization of texlets, surflings and surfaces can be found in (Mustafa et al., 2011), this section introduces only the features relevant for defining grasping actions.

Texlets

Texlets are visual features that describe the local properties of a textured surface, see Fig. 3s-i,s-ii). They store the mean color, the position and the normal of a surface patch. Although texlet features are more complex in principle, for the purpose of this paper we formalize a texlet as:

$$\Pi^T = (\mathbf{p}^T, \mathbf{n}^T, c^T) \tag{1}$$

where $\mathbf{p}^T$ is the position of the texlet, $\mathbf{n}^T$ is the normal of the surface patch, and c^T is the color of the texlet in the CIELab color space.

Initial texlets in 2D are extracted from the left image of the stereo pair (see Fig. 3s-i). The image plane is separated into local patches by applying a hexagonal grid. For cells that contain texture information, texlet features are defined by averaging appearance and disparity information over pixels in the cell. Due to the grid sampling, each texlet in 2D has up to six neighbor texlets. After constructing texlets in 3D through stereo matching (see Fig. 3s-ii), the neighboring structure is propagated from 2D to 3D. Only texlets that remain close by in 3D space, and have similar orientation and color, are labeled as neighbors in 3D.

We create links between similar neighboring texlets, i.e. texlets that are proximate, co-planar and co-color, see (Mustafa et al., 2011). These connections are propagated using the transitivity relation to derive pools of connected texlets sharing similar properties in color, position, and 3D orientation.

When creating links between neighboring texlets, due to noise it is possible that wrong connections are created. These local erroneous connections might lead to errors in the segmentation on the global level. We therefore have multiple levels of granularity in the hierarchy, so that local errors do not propagate.

Surflings

In the next level of the hierarchy, the texlets are grouped into semi-global surface descriptors called surflings, see Fig. 3s-iii. The system subdivides the pools of texlets into small subsets of about five to ten texlets using k-means clustering on the position. These subsets of similar texlets form the surfling features, which represent the mid-level abstraction bridging the texlets and the surfaces:

$$\Psi^{SL} = (\mathbf{p}^{SL}, \mathbf{o}^{SL}, \mathbf{s}^{SL}, c^{SL}, b^{SL}, \mathbf{n}_b^{SL}, \mathbf{p}_E^{SL}) \tag{2}$$

where the surfling feature is a rectangular planar patch described by a full 6D pose (position $\mathbf{p}^{SL}$, orientation $\mathbf{o}^{SL}$), size $\mathbf{s}^{SL}$ (width and length), and color c^{SL}, see Fig. 3s-iii. The boundary label $b^{SL} = (true/false)$ tells if the surfling is located at the boundary of the segmented surface. If so, the boundary normal, $\mathbf{n}_b^{SL}$, is a vector assigned to each boundary surfling, which gives the direction of the local boundary, and $\mathbf{p}_E^{SL}$ is the boundary edge point. The boundary label, boundary normal, and boundary edge-point properties will be explained at the end of this section.

The position of the surfling is the center of mass of the member texlets positions, $\mathbf{p}^{SL} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{p}^T$. The orientation is obtained using singular value decomposition (SVD) applied to the positions of the member texlets. If $\mathbf{x}^{SL}$ and $\mathbf{y}^{SL}$ are the eigenvectors with the largest and second largest eigenvalue, then the orientation of the surfling is defined as the rotation matrix $\mathbf{o}^{SL} = (\mathbf{x}^{SL}, \mathbf{y}^{SL}, \mathbf{z}^{SL})$, where $\mathbf{z}^{SL} = \mathbf{x}^{SL} \times \mathbf{y}^{SL}$. The surface of the surfling lies in the $(\mathbf{x}^{SL}\ \mathbf{y}^{SL})$-plane, while $\mathbf{z}^{SL}$ defines the surface normal. The width, $l_{\mathbf{x}}^{SL}$, and length, $l_{\mathbf{y}}^{SL}$, of the surfling are calculated based on the eigenvalues resulting from the SVD. The size in the direction i is given as $l_i^{SL} = 4 \cdot \sqrt{E_i}$, $i \in \{\mathbf{x}^{SL}, \mathbf{y}^{SL}\}$, where E_i is the eigenvalue of the corresponding eigenvector. The color property of the surfling is defined in the CIELab color space and is derived by averaging the color of the underlying texlets: $c^{SL} = \frac{1}{N}\sum_{i=1}^{N} c^T$.

Surfaces

The ECV system constructs surfaces by grouping surflings in the next level of the hierarchy, see Fig. 5b. The system establishes a neighboring structure between surflings with proximate position and orientation and performs grouping using the transitivity relation, in a similar way the surflings are created from texlets, see (Mustafa et al., 2011).

The color information is not considered when grouping surflings into surfaces, allowing for surfaces with differently colored regions. Color can be an important cue for surface segmentation and has therefore been used at the lower level to group texlets of similar color into surflings. A surface, Ψ_i^S, is defined by the set of M surflings that it contains: $\{\Psi_1^{SL}, \ldots, \Psi_M^{SL}\}$.

Once the surfaces are computed, the system identifies a subset of member surflings that are positioned on the boundaries of the surfaces. A boundary surfling, Ψ_b^{SL}, is labeled with $b^{SL} = true$ and has the additional *boundary normal* ($\mathbf{n}_b^{SL}$) property, see Fig. 5*a-i*. Figure 5*a-ii* shows a detail of the boundary of a top surface of a box. $\mathbf{n}_b^{SL}$ lies within the surface plane of the surfling and points out of the surface, it is computed as follows:

Two vectors connecting the center of a boundary surfling $\mathbf{p}_b^{SL}$ with the centers of its two closest boundary surflings $\mathbf{p}_{b1}^{SL}, \mathbf{p}_{b2}^{SL}$ are drawn: $V_1 = \mathbf{p}_{b1}^{SL} - \mathbf{p}_b^{SL}$ and $V_2 = \mathbf{p}_b^{SL} - \mathbf{p}_{b2}^{SL}$, and normalized: $V_1' = V_1/|V_1|, V_2' = V_2/|V_2|$. Their normalized sum is given with $V'' = (V_1 + V_2)/|V_1 + V_2|$, and it is further projected to the surfling's plane $V_p'' = V'' - (V'' \cdot \mathbf{z}^{SL}) \cdot \mathbf{z}^{SL}$. The normal vector $\mathbf{n}_b^{SL}$ is determined as a direction orthogonal to V_p'' lying inside the surfling plane $\mathbf{n}_b^{SL} = V_p'' \times \mathbf{z}^{SL}$, where the sign is chosen so that the normal is pointing out of the surface, see Fig. 5*a-ii*. If the angle between the lines connecting the surfling with the neighboring boundary surfling at one side and the other is bigger than a fixed threshold, we label the surfling as a corner.

For each boundary surfling that is not labeled as corner, a *boundary edge point*, $\mathbf{p}_E^{SL}$, is calculated, which is the point on the outer side of the boundary surfling, that is, on the edge of the surface, see Fig. 5*a-iii*. This is important information for the generation of grasp hypotheses, since it is the point where the finger of the robotic hand is expected to contact the object. Usually, a surface spans several surflings in width and the boundaries on each side are described with separate boundary surflings. However, when a surface has a narrow area with only one surfling in width, that surfling will support two boundaries. In that case, predicted contact points will be created on each side of the surfling, analog to the general case.

4 Grasp-Generation Methods

The hierarchical visual representation created by the ECV system as defined in Sect.3.1 provides an abstract description of the scene in terms of two hierarchies corresponding to contours and surface information. In this section, we apply this general scene representation to the problem of grasping unknown objects. By matching contours and finding surfaces of objects, we can discard many inadequate grasps and thereby drastically reduce the search space, compared to creating grasps based on more local descriptors on a lower level of the hierarchy.

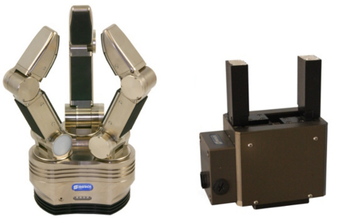


Figure 4: The two grippers used in the experimental evaluation. Left: the Schunk dexterous hand, and right: the Schunk PG70 parallel gripper

In this paper we look at three methods for generating *Elementary Grasping Actions* (EGAs) , see Fig. 1. The first method generates two-fingered grasps based on *contour* features, c_2EGAs. This method was already proposed in (Popović et al., 2010) and is briefly described in Sect. 4.1. The other two methods use *surface* features to generate two-fingered and three-finger grasps, s_2EGAs, s_3EGAs. The first surface-based method approximates the shape of the surface in a coarse way. This method is described in Sect. 4.2.1. The second surface-based method is based on the boundary surflings of the surface and is described in detail in Sect. 4.2.2.

Throughout the paper, we use the following notation for the grasp types: $[c|s]_f\text{EGA}_i$, where c denotes a contour-based grasp and s a surface-based grasp, f is the number of fingers involved in the grasp, and i is the index of the grasp subtype, where $i = 1$ for encompassing grasps, $i = 2$ for top pinch grasps, and $i = 3$ for side pinch grasp. As an example, $s_2\text{EGA}_1$ is a two-finger surface-based encompassing grasp, see Fig.1.

We choose to define a general grasp G as:

$$G = (\mathscr{C}, \mathbf{a}_d) \tag{3}$$

Where $\mathscr{C}$ is a set of two, $\mathscr{C} = \{\mathbf{c}_1, \mathbf{c}_2\}$, or three, $\mathscr{C} = \{\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3\}$ contacts, and $\mathbf{a}_d$ is the approach direction of the gripper, see Fig. 5b. A contact $\mathbf{c}$ is defined by its position, $\mathbf{c}_{pos}$, and normal, $\mathbf{c}_n$, see Fig. 5*a-iv*:

$$\mathbf{c} = (\mathbf{c}_{pos}, \mathbf{c}_n), \tag{4}$$

A "contact" defines the place and orientation that is the target for one of the gripper's fingers. For the encompassing grasps, the contacts mark the exact location of the predicted meeting between the fingers and the object. For the pinch grasps, however, the neighborhood of the detected visual feature we are trying to grasp is in general not interpreted in enough detail to estimate the exact location of the collision. We, for instance, do not reason about the thickness of an edge or a surface, or about the size of the opening between two contours. For the pinch grasp, the contacts are therefore placed so that the gripper's fingers are at a certain distance from the feature. A distance of 20 mm has been used for the experiments in this article.

In the case of surface boundary grasps (see Sect. 4.2.2), a contact region $\mathbf{c}_R$ is defined as a region of the boundary that supports the given contact point, see Fig. 5*a-iv*. This region is used to compute one of the geometric constrains for the encompassing two finger grasps. The contact region is lying in

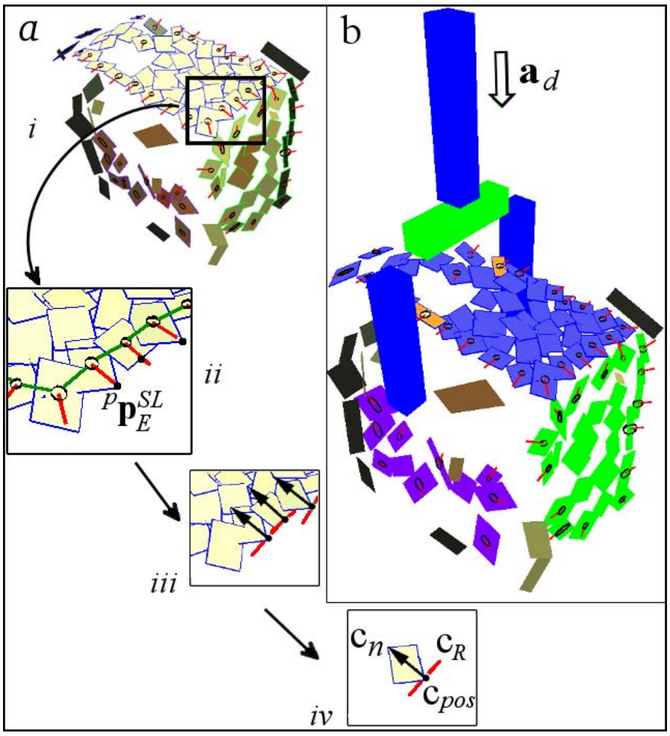


Figure 5: (a) Examples of boundary surflings. (a_i) The boundary surflings are connected by the green line, the boundary normals ($\mathbf{n}_b^{SL}$) are shown by the red line. The boundary edge points ($\mathbf{p}_E^{SL}$) are shown by the black dots and are created for the non-corner boundary surflings. (a_{ii}) Examples of contacts, given for the case of boundary surface encompassing grasps. The contact position $\mathbf{c}_{pos}$ is placed at the boundary edge point and is marked with the black dot. The black arrow shows the contact normal $\mathbf{c}_n$, and the red line shows the contact region $\mathbf{c}_R$. (a_{iii}) A single surfling and the corresponding contact, where $\mathbf{c}_{pos}$, $\mathbf{c}_n$ and $\mathbf{c}_R$ are explicitly labeled. (b) The surface segmentation has been made more explicit for the purpose of illustration. An example two-fingered grasping action is shown, which is triggered by two boundary surflings of the top surface, highlighted in orange. The approach direction ($\mathbf{a}_d$) is indicated by the arrow.

the plane of the surfling, goes through contact point $\mathbf{c}_{pos}$, and is perpendicular to the contact normal $\mathbf{c}_n$. The length of the region is given by the size ($\mathbf{s}^{SL}$) of the surfling.

The following subsections describe the computation procedures for the three grasping methods in detail.

4.1 Contour Elementary Grasping Actions – c_2EGAs

c_2EGAs are two-finger grasps built upon 3D contour features in a scene. As explained in the previous section, pairs of contours belonging to the same surface of an object have been matched. A number of grasping actions are generated for those contour pairs.

As a first step, a common plane P is fitted to the positions of line segments from both contours, see fig 6a. For each contour we produce a parametrization using non-uniform rational basis splines (NURBS) that allows for the extraction of the centers of the contours and the local directions at these centers. We project the contour centers and local directions to the common plane P. For each grasp we need to define two contacts and a gripper approach direction, as explained below.

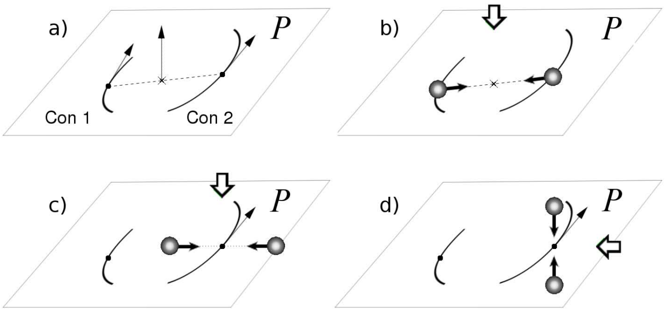


Figure 6: Contour based EGAs. a) The figure shows the two contours Con_1 and Con_2 projected to the common plane P. The centers of the contours are marked with the black dots and the corresponding vectors are marking the local directions of the contours at the center points. The vector in the middle is the normal of the common plane P. It is drawn at the center of the line connecting the contour centers. b) c_2EGAs$_1$, c) c_2EGAs$_2$, d) c_2EGAs$_3$. The spheres mark the contact positions $\mathbf{c}_{pos}$, the black vectors originating from the spheres mark the contact normals $\mathbf{c}_n$, the white arrow marks the approach direction $\mathbf{a}_d$.

c_2EGA$_1$ is an encompassing grasp created by positioning two contacts on the centers of the two contours, see Fig. 6b. The contact normals $\mathbf{c}_n$ are following the direction of the line connecting both contour centers. The approach direction $\mathbf{a}_d$ of the gripper is the inverse of the common plane normal. This type of grasp is only created when the two contours are opposite to each other and close to parallel, otherwise the object is likely to be slipped. With parallel we mean that the contours have a similar local directions at the contact points. Contacts are opposite to each other when the connecting line between the contacts is close to orthogonal to the local directions at the contact points.

c_2EGA$_2$ is a top pinch grasp assuming an opening between the two contours. The aim is to grasp one of the contours, see Fig. 6c. The contacts are positioned around the contour center, along the direction that is orthogonal both to the local contour direction and to the common plane normal. The exact positions are given by the requirement of putting one gripper finger at the center of the assumed opening. To achieve this, the contacts are placed at a distance from the to-be-grasped contour that is half the distance between the centers of both contours. The approach direction of the gripper is the inverse of the common plane normal.

c_2EGA$_3$ is a side pinch grasp that aims at grasping the surface represented by the common plane. It assumes that there is an opening just below the surface, see Fig. 6d. The contacts are positioned at either side of the contour center, along the direction of the common plane normal. As the placement of the opening below the surface is not known, the exact placement of the fingers is set to a fixed predefined value. The approach direction of the gripper is orthogonal both to the local contour direction and to the common plane normal.

For more details on the contour elementary grasping actions, we refer to (Popović et al., 2010).

4.2 Surface Elementary Grasping Actions – *s*EGAs

The textural hierarchy presented in Sect. 3.1.2 identifies surfaces of the objects in the scene. This information is used for two- and three-finger *Surface Elementary Grasping Actions*, s_2EGAs, s_3EGAs). The grasping actions belong to one of the two types shown on the bottom row of Fig. 1, where s_2EGA$_1$ and s_3EGA$_1$ are encompassing grasps and s_2EGA$_3$ is a pinch grasp. In this work, we investigate two methods for generating grasps based on surface features. The first method, discussed in Sect. 4.2.1, relies on modeling an approximate shape of the surface in a coarse way. It produces few two-finger grasps that capture the surface as a whole. The second method, introduced in (Popović et al., 2011), is explained in Sect. 4.2.2. This method is more elaborate, and operates on individual boundary surflings, which allows for a larger number of grasps and moreover for grasps that involve three fingers.

4.2.1 The SVD Method – *ss*EGAs

For each extracted surface, Ψ^S, we perform a singular value decomposition (SVD) on the 3D positions of the boundary surflings, $\mathbf{p}_b^{SL}$. This provides the center of mass and the eigenvectors, which we use to construct a local reference frame $\mathscr{L}$. The center of mass, M, is used as the origin of $\mathscr{L}$ and the eigenvectors ordered by descending eigenvalues, $\mathbf{x}, \mathbf{y}, \mathbf{z}$, form the x-, y-, and z-axis of $\mathscr{L}$, see fig 7b. The x-axis and y-axis also define a plane P fitted through the positions of the boundary surflings, while the z-axis defines the normal $\mathbf{p}_n = \mathbf{z}$ of the plane P.

In order to position the contacts, we need to estimate the dimensions of the area covered by the surface. We do so by modeling the surface as a minimum rectangle covering the boundary surflings projected to the plane P. The position of the projected boundary surflings is indicated as ${}^p\mathbf{p}^{SL}$. The rectangle is lying in the xy-plane and its orientation is aligned with the x- and y-axis. See Fig. 7c for an illustration of the estimated rectangle.

For each surface, we produce two encompassing and four pinch grasps, $G = (\{\mathbf{c}_1, \mathbf{c}_2\}, \mathbf{a}_d)$, i.e. in total six grasps of the surface along the main directions, see Figs. 7d and 8. For encompassing (s_2EGA$_1$) grasps, the contact positions $\mathbf{c}_{pos}$ are located at the centers of two opposing edges of the estimated rectangle. The contact normals $\mathbf{c}_n$ are parallel to the x or y-axis and are pointing towards the center of the rectangle. The approach direction of the gripper is the inverse of the surface normal: $\mathbf{a}_d = -\mathbf{p}_n$. For pinch grasps ($s_2EGA_3$), the contacts are surrounding the centers of the boundaries such that $\mathbf{c}_1$ and $\mathbf{c}_2$ are exactly above and below the centers of the edges (in the local coordinate frame). The height above or below the xy-plane is set to a predefined value depending on the scale of the gripper. The contact normals $\mathbf{c}_n$ are pointing vertically towards the xy-plane, aligned with $\mathbf{p}_n$. The approach directions of the gripper are parallel with either the x or y-axis, see Fig. 7d.

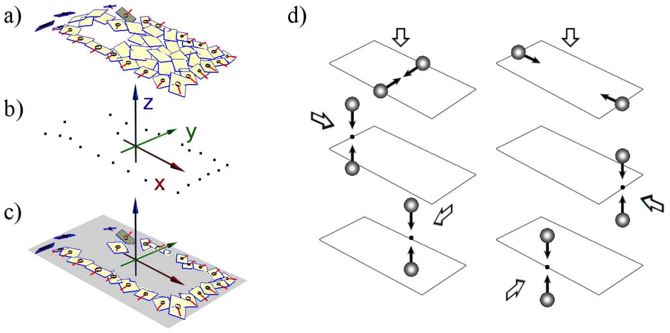


Figure 7: The SVD method explained. a) A segmented surface, boundary surflings are highlighted, b) A local reference frame is derived by singular value decomposition on the 3D position of the boundary surflings, c) A minimal rectangular covering the projected boundary surflings is used as an estimate of the size of the graspable surface. d) Six grasps along main surface directions, see also Fig. 8. The top row are the encompassing grasps. The second row are the pinch grasps along the x-axis and the bottom row are pinch grasps along the y-axis of the local reference frame.

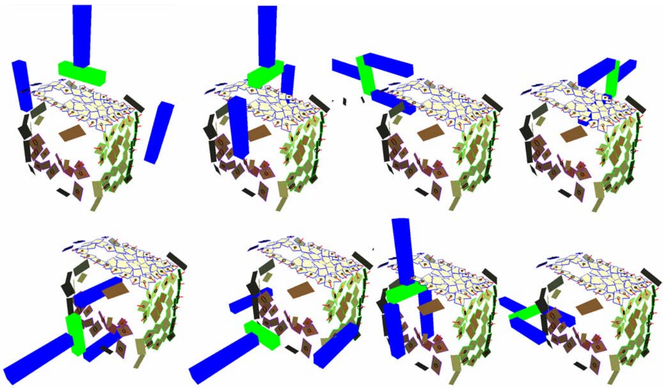


Figure 8: Examples of the SVD grasps displayed in the visualization environment. First row shows grasps for the top surface of the box from Fig. 5. Bottom row shows grasps for the side surface.

4.2.2 The Boundary Method – *sb*EGAs

The ECV system provides information about the locations of the boundaries of the surfaces. A boundary is represented in a sparse way, through properties of boundary surflings Ψ_b^{SL}, see Sect. 3.1.2 and Fig. 5. In this method we fit a plane P through the 3D positions of boundary surflings, with the plane normal $\mathbf{p}_n$, in the same way as described in the Sect. 4.2.1. We then project the boundary surflings to this plane. We notate the projected values with a 'p' in a preceding superscript, i.e., projected position ${}^p\mathbf{p}^{SL}$, projected size ${}^p\mathbf{s}_b^{SL}$, projected boundary normal ${}^p\mathbf{n}_b^{SL}$, projected boundary edge point ${}^p\mathbf{p}_E^{SL}$, and projected contact region ${}^p\mathbf{c}_R$.

By projecting surflings to a plane, we touch upon the research area of vision-based planar grasps, see e.g. (Morales et al., 2006; Chinellato et al., 2005), where objects are represented by their contours in 2D. The contours are usually derived from a single top-down view of the object and are analyzed to estimate optimal top down grasps. Morales et al. (2006) suggest numerous criteria for choosing the optimal contacts of fingers with the selected boundary regions. These cri-

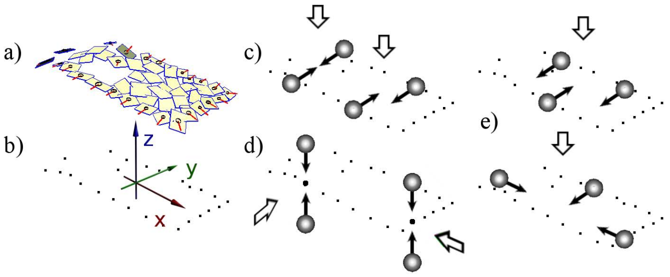


Figure 9: The Boundary method explained. a) A segmented surface, boundary surflings are highlighted, b) A local reference frame is derived by singular value decomposition on the 3D position of the boundary surflings, the black dots represent boundary edge points. c) Two examples of s_2EGA$_1$ encompassing two finger grasps. d) Two examples of s_2EGA$_3$ two finger pinch grasps. e) Two examples of s_3EGA$_1$ encompassing three finger grasps.

teria make use of parameters such as contour curvature, distribution of forces and torques, and the deviations of predicted forces and torques due to the kinematics of the robotic hands.

It is important to notice that although we make use of the principles from the planar grasp methods, we overcome some of the weaknesses connected to those methods. The purely 2D method assume the top-down view of the object, and the analyzed 2D plane is always parallel to the horizontal support surface. In our method the graspable plane originates from any surface of the object and can be in an arbitrary orientation. The second weakness of the purely planar methods is that the contour acquired from the top-down view does not inform about the height nor about the 3D shape of the object, it does not necessarily represent an existing surface of the object. In contrast, our method is based on the 3D features of the ECV representation and the grasps are aimed at the specific segmented 3D surfaces.

We now give the definition of s_2EGA$_3$ pinch grasps, followed by the definition of s_2EGA$_1$ and s_3EGA$_1$ encompassing grasps.

s_2EGA$_3$ Pinch Grasps

A pinch surface grasp, $G = (\{\mathbf{c}_1, \mathbf{c}_2\}, \mathbf{a}_d)$, is defined for each non-corner boundary surfling Ψ_b^{SL}, (see Sect. 3.1.2). A grasp is defined with a pair of contacts, where the contact positions $\mathbf{c}_{pos}$ are exactly above and below the surfling's projected boundary edge point ${}^{P}\mathbf{p}_E^{SL}$ (in the local coordinate frame), see Fig. 9d). The height above or below the plane P is set to a predefined value depending on the scale of the gripper. The contact normals $\mathbf{c}_n$ are aligned with the plane normal $\mathbf{p}_n$ and point from the contact positions towards the plane. The approach direction of the gripper is the inverse of the projected boundary normal: $\mathbf{a}_d = -{}^{P}\mathbf{n}_b^{SL}$.

The pinch grasps are defined analogous to the pinch grasps in the SVD method, see Sect. 4.2.1). The difference is that grasps are generated that target at each non-corner boundary surfling, while in the SVD method, only the four rectangular sides are grasped.

s_2EGA$_1$ and s_3EGA$_1$ Encompassing Grasps

The computation of encompassing grasps is more complex and requires several steps. Firstly, we define potential contacts connected to each non-corner boundary surfling, and secondly, we analyze combinations of two or three contact points to maintain potentially stable grasps.

A contact is defined for each non-corner boundary surfling Ψ_b^{SL}. It is placed at the projected boundary edge point: $\mathbf{c}_{pos} = {}^{P}\mathbf{p}_E^{SL}$. The contact normal is defined as the inverse of the surfling's projected boundary normal $\mathbf{c}_n = -{}^{P}\mathbf{n}_b^{SL}$, see Fig. 5*a-iii*.

Encompassing s_2EGA$_1$ and s_3EGA$_1$ grasps are based upon pairs and triplets of contact points and have an approach direction of the gripper that is the inverse of the surface normal, see Fig. 9c),e):

$$\begin{aligned} s_2\text{EGA}_1 &= (\{\mathbf{c}_1, \mathbf{c}_2\}, -\mathbf{p}_n) \\ s_3\text{EGA}_1 &= (\{\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3\}, -\mathbf{p}_n) \end{aligned}$$

A typical surface will potentially have many contacts. Not all of the possible combinations into pairs and triplets of contacts will represent a valid grasp. Hence a task in the grasp-generating procedure is to efficiently detect suitable contact combinations, such that associated actions are likely to result in a stable grasp that can resist forces and torques produced by gravity and other external disturbances.

The selection criteria are based on gripper-specific kinematic constraints and on geometric constraints. We do not apply the commonly used grasp-stability measures based on the wrench space (Miller and Allen, 2004) because these measures usually assume perfect knowledge of the object's shape. Since we are dealing with visual observations of unknown objects in the real world, the derived shape representation will be somewhat noisy and uncertain. We therefore apply less detailed heuristics instead, with the purpose of maximizing the grasp stability.

We apply the constraints in the order of increased computational complexity. The first constraint is the gripper-specific constraint that filters out the contact combinations that are too far apart, having in mind the maximal distance between the gripper fingers. We apply two additional geometric constraints both for two finger and three finger grasps.

The geometric constraints aim to prevent sliding and to minimize the torques. As one of the geometric constraints, we use the Coulomb's friction model $F_f \leq \mu \cdot F_n$ to derive a friction cone for each contact. The friction coefficient μ defines a friction half angle: $\beta = \arctan(\mu)$, that is the maximum distance in angle between the contact normal and the direction of the force applied by the finger, at which sliding will not occur, see Fig. 10a). Since the friction coefficient is not known, we empirically set it to a conservative value of 0.3.

In the case of contact pairs, s_2EGA$_1$, we first require that the angle between the contact normals is within $\mathbf{n}_1 \cdot (-\mathbf{n}_2) < \beta$, see Fig. 10b). Additionally we project the contact regions $\mathbf{c}_R$ in the direction of the contact normals, and demand that at least one of the projected regions intersects with the opposite contact's region, see Fig. 10c).

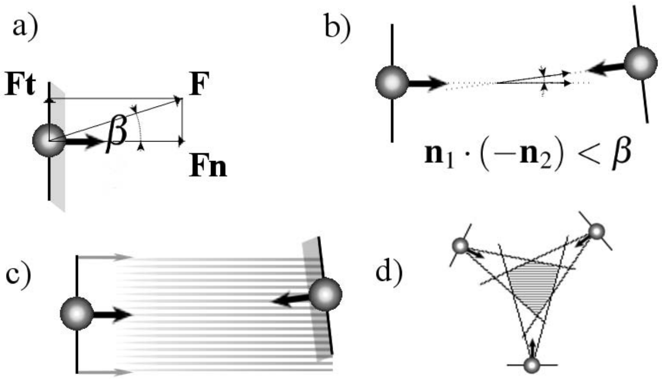


Figure 10: a) Friction half angle. b) The constraint on mutual orientation for contact pairs. c) The constraint on the mutual position for contact pairs for two finger grasps. d) The constraint on the mutual position for contact pairs for three finger grasps.

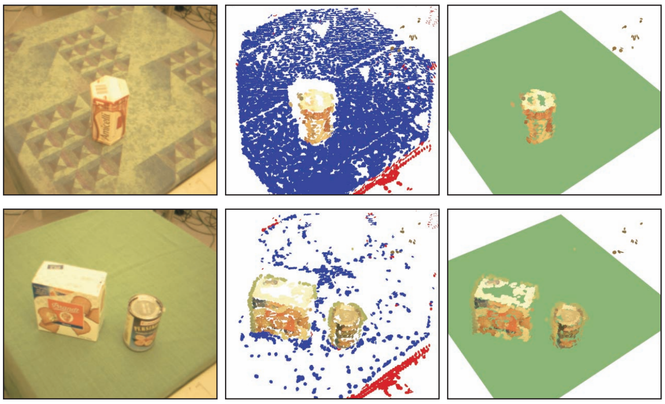


Figure 12: Detection of the supporting plane. The left column shows the original left camera image. The middle column shows the features belonging to the table plane in blue, those below the plane in red, and the features above the table plane are given in their original color. The right column shows the remaining features after filtering and the estimated plane model.

For triplets of contacts, $s_3\mathrm{EGA}_1$, we first require that the contact normals positively span the plane. This is the case if each of the three vectors can be written as a linear combination of the other two using only positive weights. Second, we require that the intersection of friction cones is not empty (Ponce and Faverjon, 1995), see Fig. 10d).

4.3 Filtering of Features based on Supporting Plane Detection

The ECV scene representation contains visual features of all elements in the scene. For the grasping purposes, we are only interested in the features of the objects that are placed on the table in front of the robot. To filter out all irrelevant features, we detect the dominant supporting plane in the scene and keep those features that are above that plane.

All features that are more than 10 mm above the table plane are kept and the rest is filtered out. Examples of this filtering step are given in Fig. 12. The first column shows the original left camera image. The middle column illustrates the plane detection. In blue are all the features that belong to the detected supporting plane, red features are those below the plane, and the remaining features above the plane are displayed in their original color. The last column shows the estimated plane in green with the remaining features. This information is used to predict collisions, as is described in Sect. 5.2.

5 Grasps Execution

This section describes how the grasps are executed. We first describe how the robot hand configurations are derived from the grasp definitions in Sect. 5.1. Section 5.2 explains how collision detection is used to prevent grasp configurations that are predicted to collide with objects in the scene. The execution of the grasps is described in Sect. 5.3. Finally, we explain the evaluation of the grasps in Sect. 5.4.

5.1 Inverse Kinematics

Once a general grasp has been defined through the set of contacts and the approach direction, (see Eq. 3), the inverse kinematics is used to search for the feasible gripper configurations that will place the gripper fingers at or close to the desired locations. The inverse kinematics makes the following mapping:

$$G = (\{\mathbf{c}_1, \ldots, \mathbf{c}_n\}, \mathbf{a}_d) \mapsto (\mathbf{X}_{\text{hand}}, \mathbf{q})$$

where $\mathbf{X}_{\text{hand}}$ is the 6-dimensional Cartesian pose of the hand base, and $\mathbf{q}$ is the joint configuration. For the parallel jaw gripper, $\mathbf{q}$ is a one dimensional vector marking the distance between the fingers. The Schunk dexterous hand has seven degrees of freedom, and $\mathbf{q} = \{q_0, \ldots, q_6\}$ therefore gives the angles of the seven joints.

The grasps are defined by the contact positions at the detected features (contours or surface boundaries), which define the extremities of an object. In order to generate stable grasps, the end effectors of the gripper need to go beyond those points and grasp with a certain depth in the approach direction to have the fingers get a better grip on the object. We do this by translating the hand pose in direction of the approach vector:

$$\mathbf{X}'_{\text{hand}} = \mathbf{X}_{\text{hand}} + d \cdot \mathbf{a}_d$$

where d is the depth of a grasp. In our experiments, we use $d = 20$ mm both in simulation and in real experiments.

For evaluating grasps in the real setups, we need to compute the inverse kinematics of the robot arm. i.e. find the configuration of the robot arm $\mathbf{Q}_{\text{arm}}$ that will place the robot hand in the desired $\mathbf{X}'_{\text{hand}}$. We also make sure that the final hand configuration is accessed from the approach direction, and thus define an additional hand configuration $\mathbf{X}^{\text{a}}_{\text{hand}}$ that is to be accessed before the final $\mathbf{X}'_{\text{hand}}$:

$$\mathbf{X}^{\text{a}}_{\text{hand}} = \mathbf{X}'_{\text{hand}} - d^{\text{a}} \cdot \mathbf{a}_d,$$

and search for the corresponding $\mathbf{Q}^{\text{a}}_{\text{arm}}$. The grasps for which either of the $\{\mathbf{Q}_{\text{arm}}, \mathbf{Q}^{\text{a}}_{\text{arm}}\}$ can not be found are discarded. In our experiments, we use $d^{\text{a}} = 30$ mm both in simulation and in real experiments.

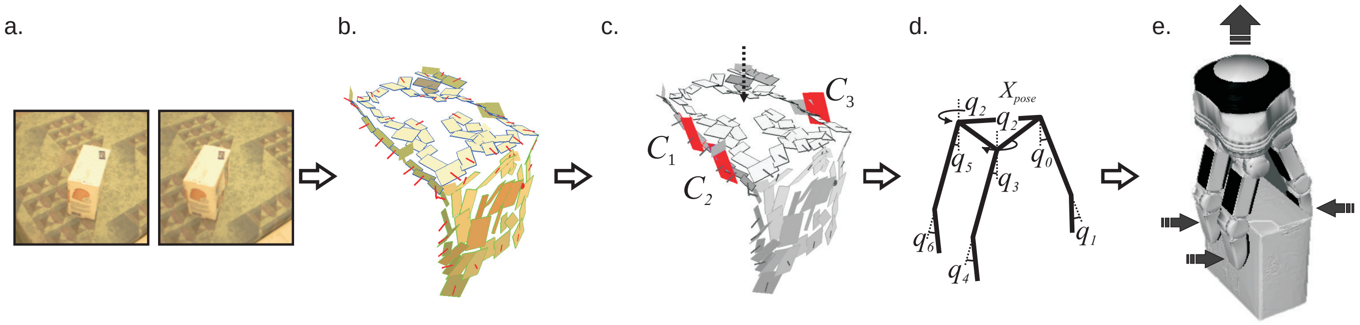


Figure 11: The mixed real-world and simulated experimental setup.

5.2 Collision Detection

Our methods do not consider all global information, but rather generate grasps based on surface information. This potentially can result in grasps where the robot would be penetrating other objects in the scene. To prevent this, we predict collision of the robot and elements in the scene using the models of the robot and the hand, and the global scene representation illustrated in the right column of Fig. 12, where the supporting plane is estimated as described in Sect. 4.3. When in the pre-grasping stage, the robot is expected to collide with or penetrate the supporting plane, or any of the scene features, the grasp hypotheses in question is discarded.

5.3 Grasp Execution

In order to execute a grasp, the system has to perform preparatory movements. The gripper is first set to a pre-grasp configuration $\mathbf{q}_{\text{open}}$, where the gripper is opened more than $\mathbf{q}$, and the base of the hand is positioned at the approach pose $\mathbf{X}^{\text{a}}_{\text{hand}}$, which is a distance d^{a} away from the target hand pose $\mathbf{X}'_{\text{hand}}$. Next, the hand is moved to the target pose $\mathbf{X}'_{hand}$ and the fingers begin to close. In the closing of the fingers, the grasp-control policy guides the joint configuration from $\mathbf{q}_{\text{open}}$ to $\mathbf{q}_{\text{closed}}$, where the gripper is closed more than $\mathbf{q}$. The grasping action is finished either when the joints settle in a static configuration, or when the joint configuration $\mathbf{q}_{\text{closed}}$ is reached.

For the parallel jaw gripper, $\mathbf{q}_{\text{open}} = \mathbf{q} + a$, and $\mathbf{q}_{\text{closed}} = 0$, that is, in the opening configuration, the fingers are $a = 10$ mm wider. For the Schunk dexterous hand, the joints at the base of the fingers are changed in the opening and closing configurations: $\mathbf{q}_{open} = \mathbf{q} + \{\alpha, 0, 0, \alpha, 0, \alpha, 0\}$ and $\mathbf{q}_{closed} = \mathbf{q} + \{\beta, 0, 0, \beta, 0, \beta, 0\}$. In our experiments, we used $\alpha = -0.5$ rad in case of one object, $\alpha = -0.3$ rad for multiple objects, and $\beta = 0.2$ rad. Note that in case of the top pinch grasps, c_2EGA$_2$, $a = 0$ and $\alpha = 0$, because one of the fingers is targeted right in between the two contours.

5.4 Grasp Evaluation

The object is grasped when, after the grasp action is finished, all fingers are in contact with the object. However, that does not mean that the grasp is stable. In order to test the grasp quality, we perform a dynamic evaluation of the grasp by lifting the object and observing the outcome.

In the real-world experimental setup, a grasp is evaluated as successful when the object is still in the hand after lifting it. In case that the object drops from the gripper, is not grasped in the first place, or a collision occurred, the grasp is evaluated as failed. More details are given in Sect. 6.2 and Sect. 6.3.

The mixed real-world and simulated experimental setup allows us to make a more elaborated evaluation. Apart from classifying the grasps as *stable*, *slipped*, *dropped*, or *missed*, we get a continuous lift quality measure by observing how much the object slipped from the hand during lifting. We furthermore calculate the static grasp wrench-space measure. Details on these measures are given in Sect. 6.1.

6 Experiments

The grasp-generation methods have been tested in two different experimental setups. First, experiments have been performed in a mixed real-world and simulated setup, (in following also called the hybrid setup), using the VisGraB benchmark (Kootstra et al., submitted), see Sect.6.1. The benchmark provides stereo images of a large set of real scenes, and the grasps are performed using a dynamics based grasp simulator. This enables extensive testing on real visual input and allows for a comparison between methods under the exact same circumstances. Second, experiments have been performed in a real-world setup to demonstrate the performance of the methods running on real robotic systems. Here two scenarios have been used, one using a parallel jaw gripper described in Sect.6.2, and one using a three-finger dexterous hand described in Sect.6.3.

6.1 Hybrid Real-World and Simulated Setup

The methods have been tested on the VisGraB benchmark[1], which we have presented in Kootstra et al. (submitted). The benchmark provides a hybrid real-world and simulated experimental setup. Real camera images provide the input to the ECV system and the grasp-generation methods. The produced grasp hypotheses are then tested in a dynamic simulated envi-

[1] http://www.csc.kth.se/visgrab

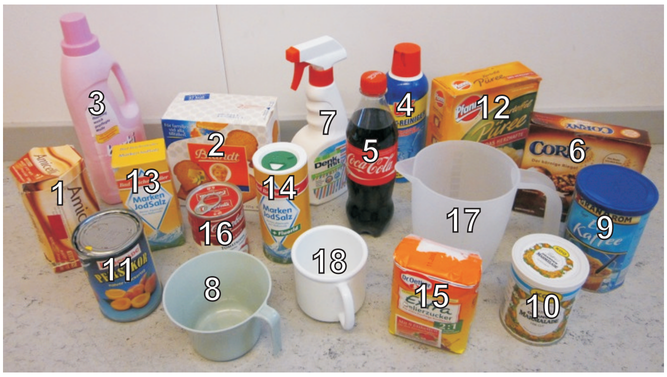


Figure 13: The 18 objects used in the hybrid experimental setup. The figure is taken from (Kootstra et al., submitted).

ronment[2]. Figure 11 gives an illustration of the hybrid real-world and simulated setup. This setup gives us the possibility to run a large number of trials and to repeat the experiments in the exact same conditions, allowing for a fair comparison among methods, while having to deal with the noise and uncertainty of the real world. A total of 47,269 grasps have been performed in our experiment.

Real stereo-camera images of a large number of grasp scenes are provided by the benchmark. The 18 different objects used are shown in Fig. 13. The objects have various shapes, sizes, colors, and textures. The benchmark contains scenes with one object and scenes with two objects. The single-object scenes contain the 18 different objects in eight different poses, four where the object stands upright, and four where the object lies down. In the two-object scenes, 9 combinations of objects are included, where the objects are in eight different configurations, four with the objects placed apart, and four with the objects touching each other. All scenes are recorded in two conditions, placed on a non-textured and on a cluttered/textured table. This gives in total $2 \times (18 \times 8 + 9 \times 8) = 432$ scenes. Some example scenes are given in Fig. 14.

The VisGraB benchmark furthermore contains simulated versions of all the real scenes in order to test the grasp performance in simulation. The objects and models that are used are part of the KIT Object-Models Web Database[3]. These models have been obtained using a laser scanner and therefore provide a realistic representation of the scene. Besides the objects, also the table has been placed in the simulated scene. Figure 14 (bottom row) gives some examples of simulated scenes.

The grasps are evaluated in simulation using RobWork[4]. RobWork is a simulator for robotic grasping with dynamic capabilities, which has been used in several related experiments (Jørgensen et al., 2010). The simulator has been shown to be very realistic in (Ellekilde and Jørgensen, 2011), where several thousands of grasps with a parallel gripper in a real robotic setup have been compared to the simulation. In the experiments, we use a simulation of the Schunk dexterous hand, which allows for both two-fingered parallel grasping and flexible three-fingered grasping, see Fig. 4.

Using a dynamic simulator allows us to not only look at static quality measures of the grasp, but also to determine the actual grasp success by observing the dynamical and physical consequences of grasping and lifting the object. In an experimental trial, the quality of the generated grasp is tested as follows: the hand is placed in the determined pose X'_{hand} and the fingers are opened in the opening configuration, $\mathbf{q}_{\text{open}}$. The fingers then close to the closing configuration, $\mathbf{q}_{\text{close}}$. The object is potentially grasped when the hand settles in a static configuration and the fingers touch the object. However, this does not necessary mean that the grasp is stable. To test the stability of the grasp, the hand attempts to lift the object. The result is classified as follows:

Stable grasp The object was grasped and held after lifting, with little or no slippage of the object in the hand.

Object slipped The object was grasped and held after lifting, but there was considerable slippage of the object in the hand.

Object dropped The object was grasped, but after lifting, the object was no longer held by the hand.

Object missed The object was not grasped by the hand. This is the case when the fingers are not in contact with the object after the grasping action has finished.

In collision The initial hand configuration produced a situation where the hand was penetrating the object(s) and/or the table.

The grasp is considered successful when the returned result is either *object slipped* or *stable grasp*. In both cases, the object is held in the hand after lifting.

In addition, the benchmark provides two grasp-quality measures: 1) the lift quality, $Q_{\text{lift}} \in [0,1]$, which is a dynamic measure of the grasp quality, inversely related to how much the object moves with respect to the hand during lifting, and 2) the grasp wrench-space measure, $Q_{\text{GWS}} \in [0,1]$, which is a static measure of stability. For more information, we refer to Kootstra et al. (submitted).

All three grasp-generation methods introduced in Sect. 4 have been tested on the complete benchmark. The results are given in Sect. 7.1.

6.2 Real-World Setup – Parallel Jaw Gripper

Fig. 15 gives an overview of the real-world setup using the parallel jaw gripper. The setup consists of a Bumblebee2 color stereo camera, an industrial six degrees of freedom Staubli RX60 robot arm, a Schunk PG70 2-Finger Parallel Gripper, and a force-torque sensor that is mounted between the robot's wrist and the gripper in order to detect collisions.

The nine objects used in the experiments are shown in Fig. 16. Experiments have been performed both with nine scenes containing a single object and with 56 cluttered scenes

[2] A movie illustrating the hybrid real-world and simulated setup is available at http://covil.mmmi.sdu.dk/videos/visgrabCompressed.mp4

[3] http://wwwiaim.ira.uka.de/ObjectModels

[4] http://www.robwork.org

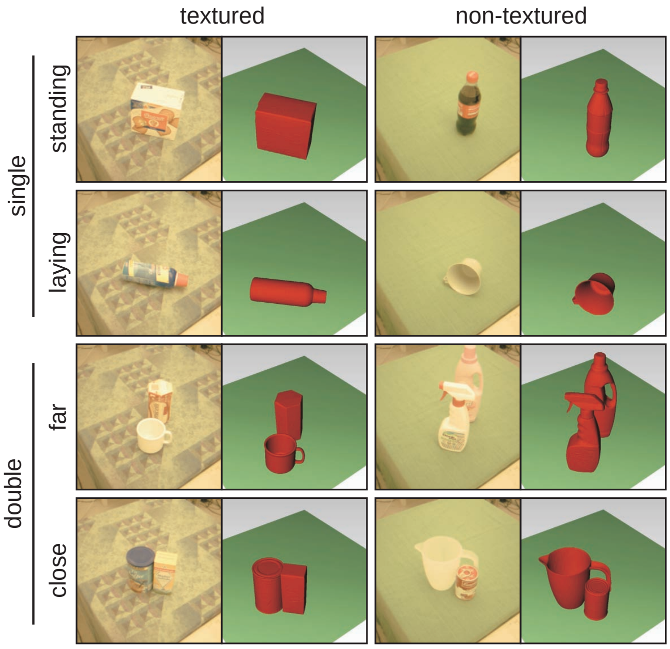


Figure 14: Example scenes of all conditions included in VisGraB. The real camera images (left image of the stereo pair) are shown on the left and views on the corresponding modeled scenes used for grasp simulation are given on the right.

containing three objects. All three grasp-generation methods introduced in Sect. 4 have been tested. From the set of suggested grasping hypotheses we select one hypotheses of a given method at random.

We performed 258 grasping trials with the single-object scenes, approximately 10 trials per method, per object. The objects were placed in a random orientation for each grasping attempt, with the similar pose of the objects for the three different methods.

For the cluttered scene, we performed 168 grasping trials. The 56 different scenes were reproduced three times, once for every method, see Fig. 17. The grasp-generation methods produces grasps for the whole scene. Certain objects will trigger more elementary grasping actions then others. Since the performed grasps are selected at random, grasps for those objects are attempted more than for the other objects.

The outcome of a grasp is labeled as successful when the object is held in the hand after lifting. If the object is dropped, missed, of if a collision occurred, the grasp is labeled as failed. The results of these experiments are given in Sect. 7.2.

6.3 Real-World Setup – Three-Finger Dexterous Hand

The hardware setup used for the experiments with the three finger hand consists of a six degrees of freedom industrial robot arm Universal Robot (UR5), a static Bumblebee2 color stereo camera, a force torque sensor mounted at the robot's wrist and a Schunk Dexterous Hand mounted on the Force Torque sensor, see Fig. 18. The floor is covered with flexible foam layer.

The six objects used in the experiments are shown in Fig. 19.

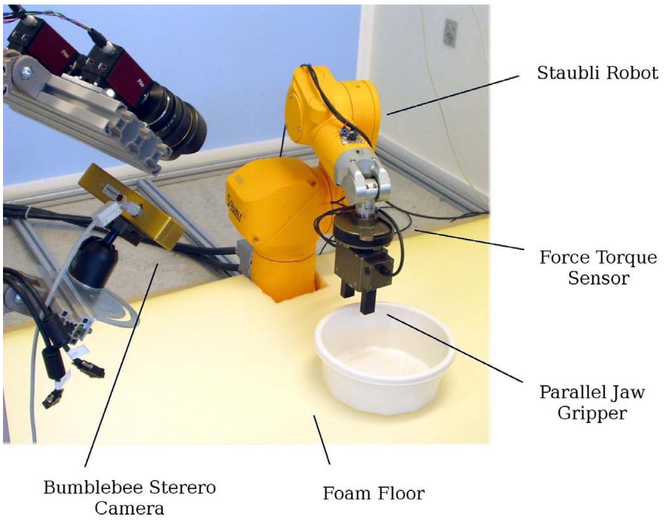


Figure 15: Experimental setup with the parallel jaw gripper.

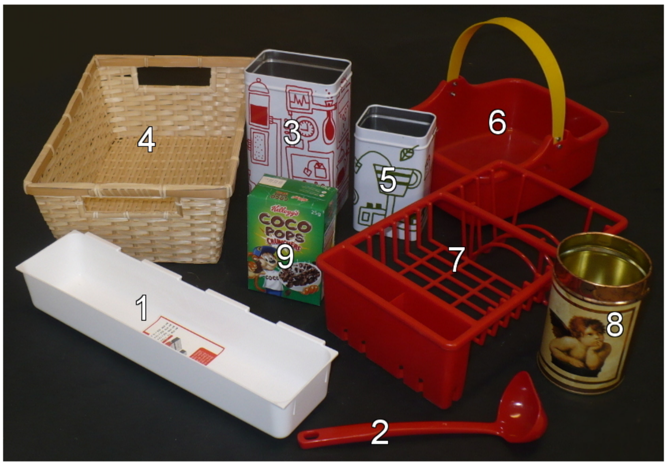


Figure 16: The nine objects used in the real-world setup - parallel jaw gripper.

Objects 1,2 and 5 are the objects from the KIT database, also used in the hybrid real-world and simulated setup. Objects 3,4 and 6 are objects also used in the real-world setup with the parallel jaw gripper.

In this setup the different grasping methods have been tested, including three two-fingered and one three-fingered grasping method. We performed 120 grasping trials for single-objects. Five different scenes were produced for each object, three where the object was standing up and 2 where the object was laying down, see Fig. 20. Each scene was reproduced four times, once for each grasping method.

For the two-object scenes we tested 72 grasps. The six objects were grouped in three pairs, and each pair was tested withing six scenes, where in three scenes objects were close together and in the remaining scenes objects were far apart. As in the single-object case, each scene was reproduced four times, once for each grasping method.

The results of these experiments are given in Sect. 7.3.

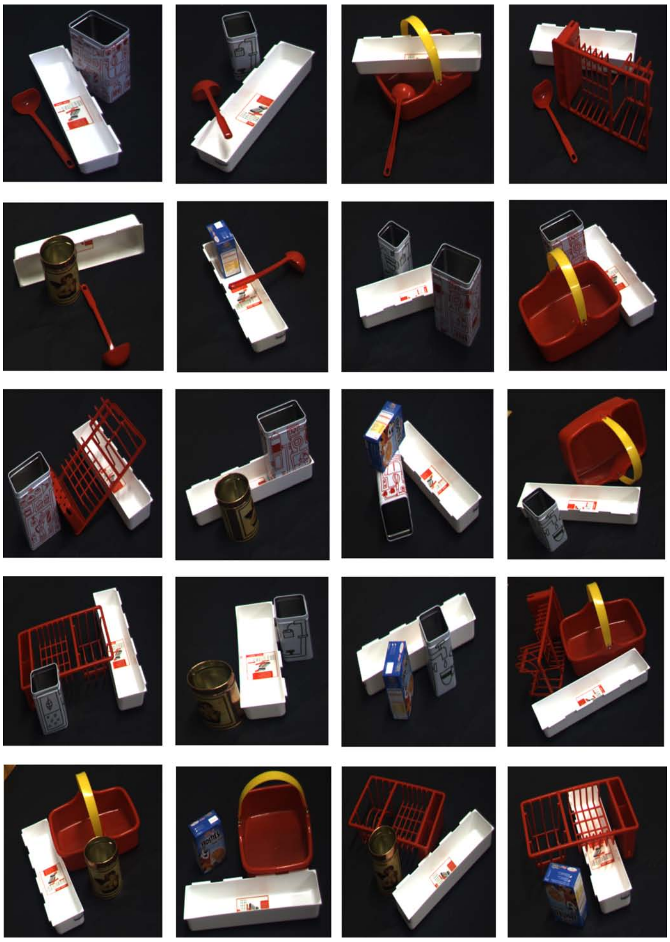


Figure 17: Examples of cluttered scenes, each containing three objects.

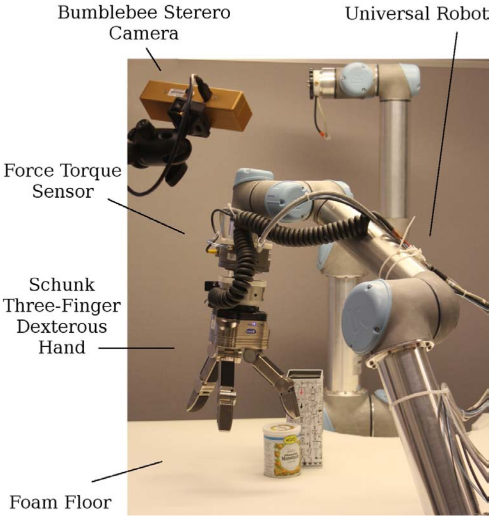


Figure 18: Experimental setup with the three-finger dexterous hand.

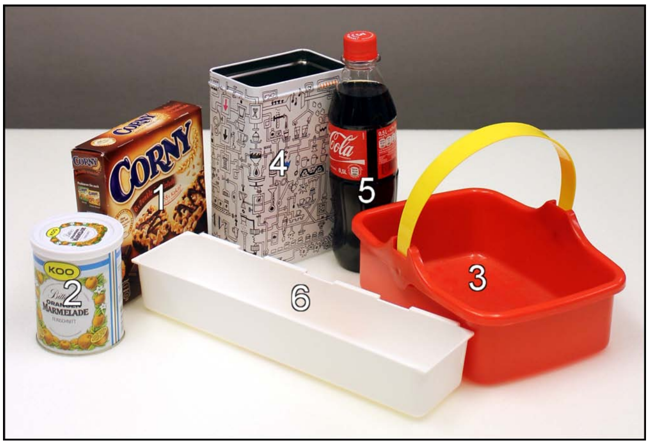


Figure 19: The six objects used in the real-world setup with the three-finger dexterous hand.

7 Results

The results of the experiments are given below. We also discuss and compare the results from the different setups and scenarios. The hybrid setup gives the possibility to test a large number of grasps. It also gives the possibility to test all the suggested grasps for one scene. In the case of the real-world experiments, only one of the suggested grasps can be tested per scene since in general the scene is disturbed by the performed grasp attempt. The large number of experiments performed in the hybrid setup provides the in-depth view of the performance of grasping methods and their sub-types, while the evaluation from the real-world experiments discriminates only between the general grasping methods.

7.1 Hybrid Real-World and Simulated Experimental Setup

7.1.1 Distribution of Grasp Results

Fig. 21 shows the grasp results for the hybrid setup, introduced in Sect.6.1. The bar plots give the distributions of all grasps averaged over the scenes. The results are split up for the different conditions: single objects standing up, single objects laying down, two objects far apart, and two objects close together. In general, it can be seen that the surface-based grasp methods perform better than the contour-based methods. The encompassing grasps, i.e., EGA_1 grasps, are more successful than the pinch grasps. The three-finger surface-boundary grasp, $sb_3\mathrm{EGA}_1$ outperforms the two-finger surface-boundary grasp, $sb_2\mathrm{EGA}_1$. The surface grasps based on the SVD method, ssEGA, perform similarly to those based on the boundary method.

In general, all grasping methods perform better when the single objects are standing upright than when they are laying down. For the double-object scenes, the amount of successful grasps is similar for the objects apart or close together, but in the later case, there is a higher proportion of collisions.

On a large proportion of scenes, the contour-based encompassing grasp, $c_2\mathrm{EGA}_1$, does not suggest any grasps. This is due to the low number of detected contours and the strict requirements of two contours to be parallel and the contact points to be opposing.

For all conditions, the side pinch grasps, i.e., EGA_3 grasps,

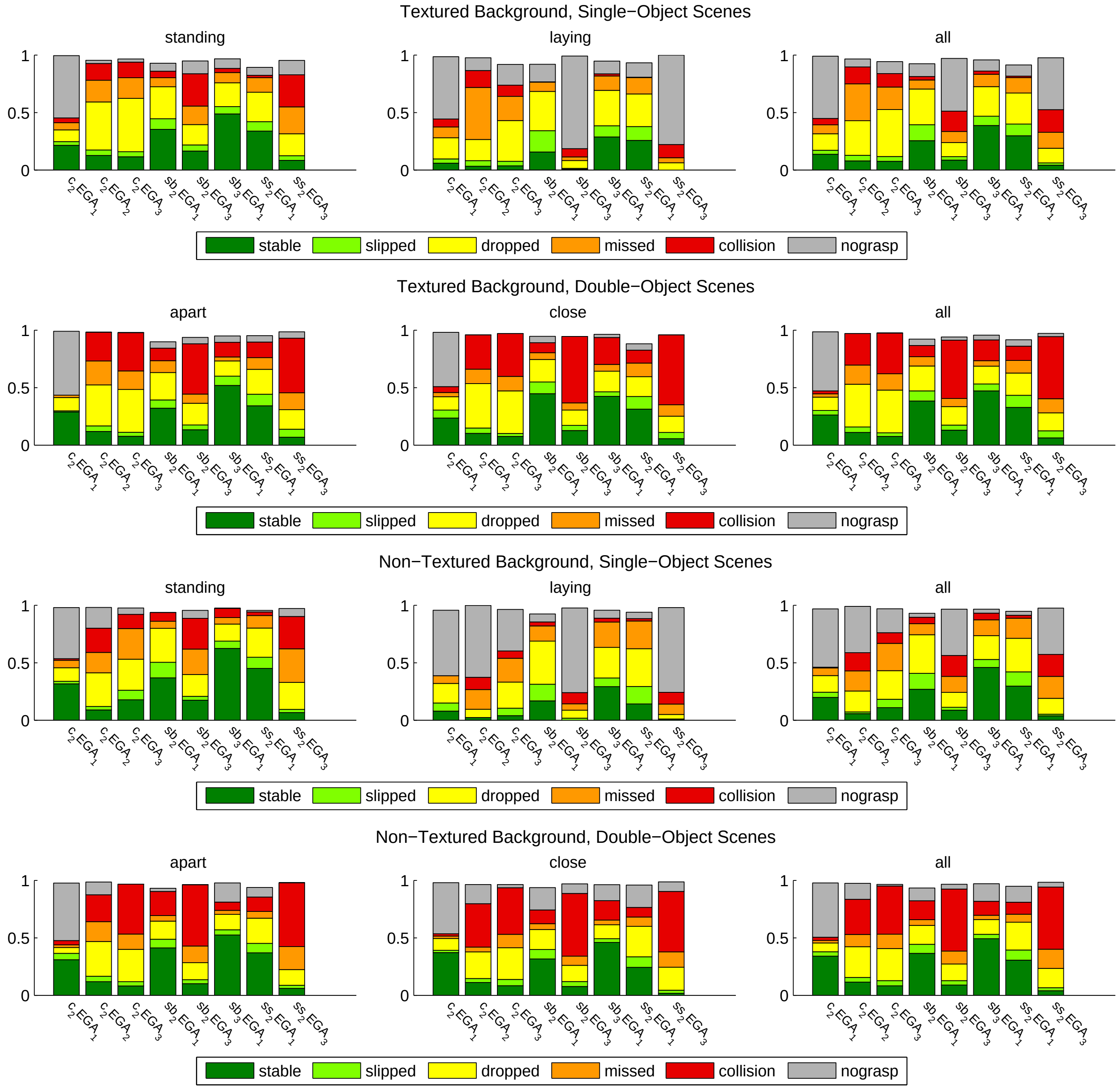


Figure 21: Grasp results for the hybrid real-world and simulated experiments. The stacked-bar plots show the average distribution of all grasps over all scenes. The stable and slipped grasps are considered successful grasps, where the object is held in the hand after lifting. The gray area shows the proportion of scenes where the methods do not suggest any grasps.

result in more collisions compared to the related encompassing grasps. This is caused by the lack of structural information at the backsides of the object due to self occlusions, which has the consequence that the grasps are not filtered out by the collision-detection mechanism. This is especially the case for the double-object scenes. For the laying objects, however, we do correctly filter out most of the pinch grasps, leading to the large number of scenes where no grasps are suggested. This is because the height of the objects is less when laying down, which reduces the size of the self-occluded part of the object, making collision detection more successful.

7.1.2 Grasp Success as a Function of the Number of Grasp Attempts

The average grasp success rates are somewhere between 0.3 and 0.6. This means that the robot will often not be able to grasp an object at the first attempt. We want to stress that this is a remarkably high performance since the system does not make use of any object knowledge. Moreover, the robot can

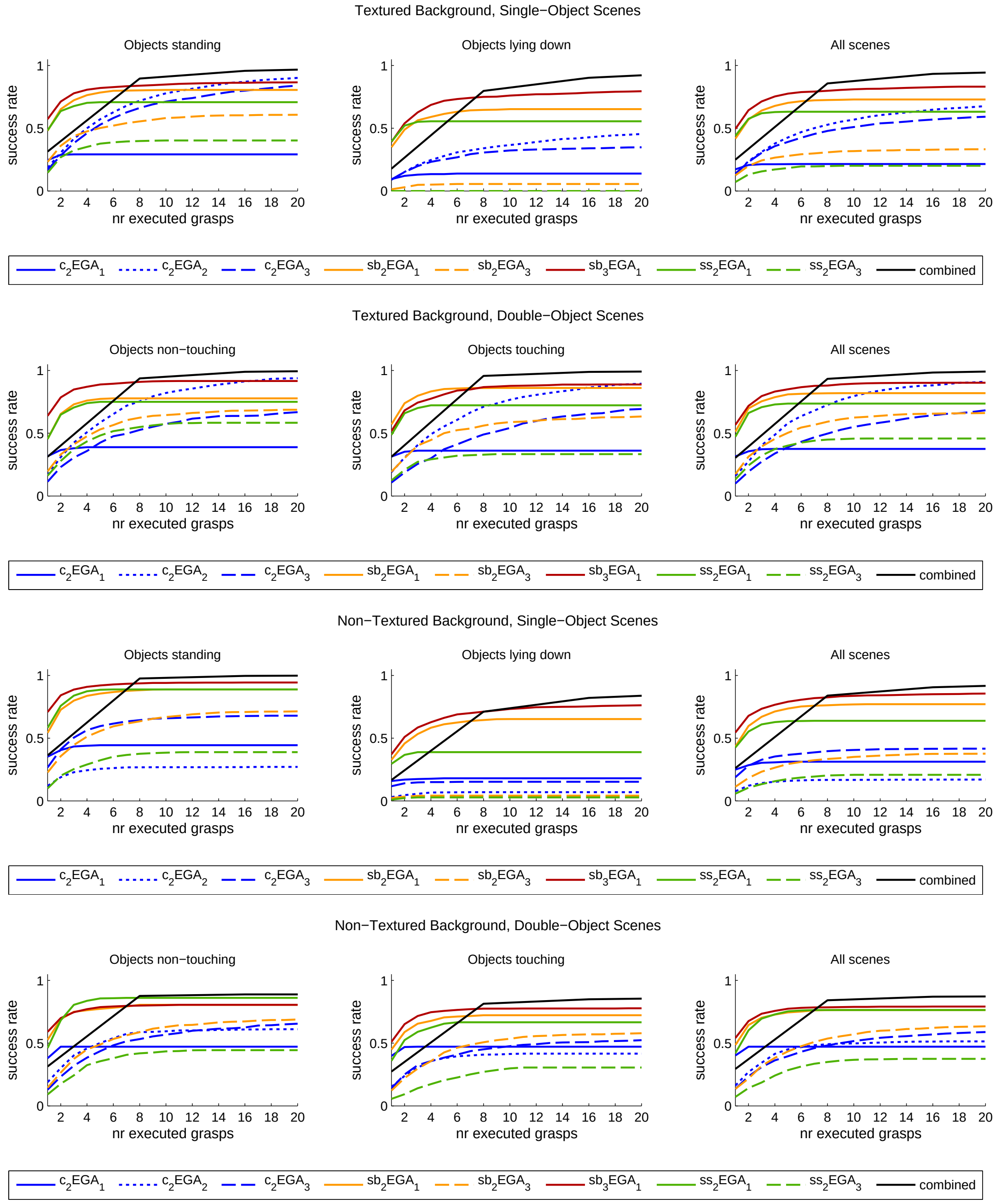


Figure 22: Grasp results for the hybrid real-world and simulated experiments. The plots show the average success rate as a function of the number of grasp attempts for the different grasp methods. The black line shows the performance when the different methods are combined. The fact that the combined results surpass the results of the individual methods shows that they are complimentary.

try another grasp if the first one fails. Fig. 22 shows the grasp success as a function of the number of grasp attempts, N. For a given scene, N grasps are taken at random from the set of generated grasps. We define it as success if any of these grasps is successful. The plots also show the combined success rates, where the N grasps of the eight different elementary grasping actions are taken together. The plots show the average success rate for all scenes over 20 randomized runs. With the combined success rates we investigate if the different grasp types are complementary, i.e. if using more then one grasp type will increase the success rates.

All curves indicate that the success rate increases when multiple grasps are attempted. The steeper the slope, the more successful additional attempts can be employed. Especially of interest is the black solid line, which shows the grasp success when the different elementary grasping actions are combined. For a single attempted grasp, the success rate is the average of the individual method. However, when one grasp of all eight methods are combined (at $N = 8$), the curve surpasses the curves of the individual methods in all cases. This shows that

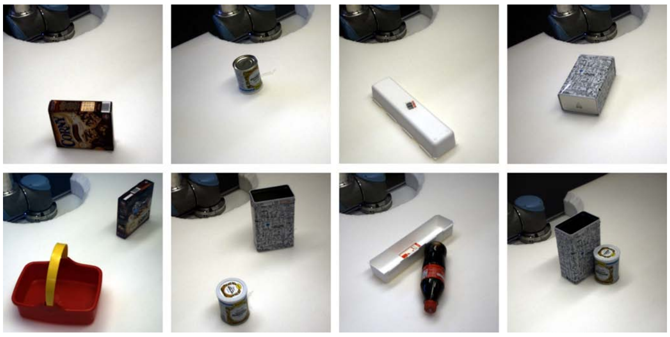


Figure 20: Examples of experimental scenes for the real-world setup with the three-finger dexterous hand.

the different methods are complimentary and that our total proposed grasp method benefits from having different elementary grasping actions based on different types of visual information.

The plots in Fig. 22 also confirm some of the results given in Sect. 7.1.1. The surface-based grasp (orange, red, and green) outperform the contour-based grasps (blue). For the surface-based grasps, the encompassing grasps (solid lines) have a higher success rate than the encompassing grasps (dashed and dotted). The contour-based encompassing grasp (solid blue), however, gains little from multiple grasp attempts, which is caused by the low number of grasp suggestions. The three-finger contour-boundary encompassing grasp (solid red) outperforms its two-finger counterpart (solid orange) and is the most successful grasp type overall.

Although for the first grasp attempt, the surface grasps based on the SVD method (green) have a similar success rate as those based on the boundary method (orange), the later is superior when more grasps are attempted. This can partly be explained by the larger number of suggested grasps using the boundary method.

Despite having lower average scores, the contour-based method contributes to the overall system by complimenting the surface-based methods when objects are low textured and surface hypotheses based on texlets and surflings cannot be made.

Given the sparse representations of the scene and the heuristics for grasp selection, the grasp-generation methods suggest only a small number of grasps. On average 5-20 grasps are generated per scene for the different methods. And as can be seen in Fig. 22 good grasp results are generally achieved already after a few attempts, especially when the different grasp methods are combined. The combination of sparseness, complementarity, and high performance is the main contribution of our method.

7.2 Real-World Setup – Parallel Jaw Gripper

Results of the real-world setup – parallel jaw gripper experiments are shown in Fig. 23. The figure gives the average success rates per object and grasping strategy, and illustrates the supplementary nature of the different grasping methods. The top figure shows the results for the single-object experiments, and the bottom figure shows the results for the cluttered scenes experiments.

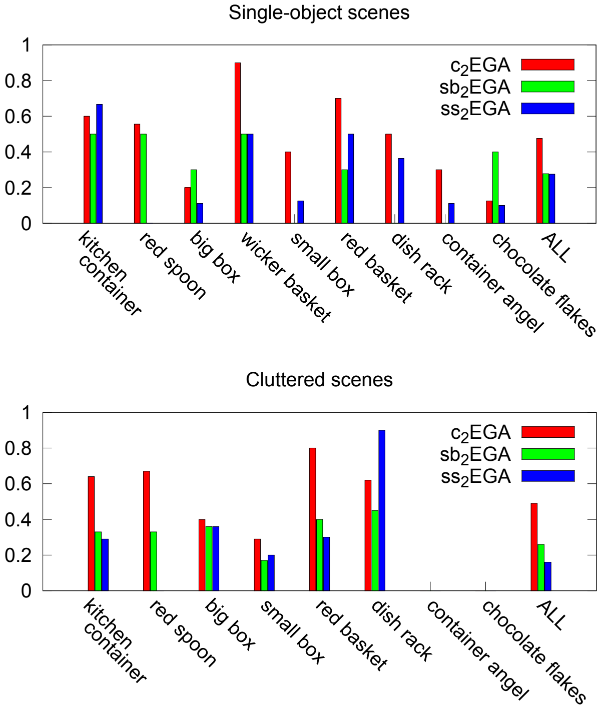


Figure 23: Grasp results for the real-world setup – parallel jaw gripper experiments. Histograms show success rates for different grasping methods and different objects, and an averaged success rate of different methods for all objects. Top figure gives results for the single-object scenes and the bottom figure gives results for the cluttered scenes shown in Fig. 17. The average number of experiments per one method and one object is 9 for single-object scenes, and 7 for the cluttered scenes. In case of cluttered scenes this numbers varies as lot and in some cases only few grasps are tested.

Both results from the single-object and cluttered scenes indicate that the contour method is the most successful method in this scenario. This seems to contradict the results from the previous Sect. 7.1 and can be explained with the fact that due to the small gripper size, most objects chosen for this scenario posses an opening (Fig. 16), and can often be grasped only by using a pinch grasps on the sides of the opening. The contours extracted from the top surface of an object often suggest good grasps. In contrast, the majority of the KIT objects used in the hybrid setup (Fig. 13) do not have an opening and can not accommodate pinch grasps. In the real parallel jaw scenario the gripper does not perform a great number of encompassing grasps as the maximal distance between fingers is about 68 mm, which means that the graspable objects can not be bigger then 50mm.

The fact that the surface grasps perform worse than contour grasps can also be explained with fewer textured objects in this scenario, compared to the objects used in the hybrid setup. It can be observed both here and in the following Sect. 7.3 that for the weakly textured objects, the contour-based grasps give better results.

For two of the objects in the cluttered scenes no successful grasps were produced. This is due to the relatively small number of grasp trials for these objects, in average three to four trials per object. Since a random grasp is selected for each of the cluttered scenes, some objects were grasped less often than others.

In general, the experiments show an overall performance comparable to the results gained from the simulation. The system is able to successfully grasp unknown objects, both for the single-object and for the cluttered scenes, using the two-finger parallel jaw gripper. In order to test and use all possible grasps for the given size of objects, a bigger gripper would be needed.

7.3 Real-World Setup – Three-Finger Dexterous Hand

For this scenario we have chosen three objects used in the hybrid setup and three objects from the real parallel jaw scenario, see Fig. 19. Both well-textured and weakly-textured objects are represented, as well as open and closed objects. The Schunk three-finger dexterous hand matches well the size of the objects and is able to perform both two-finger and three-finger grasps. By using the same hand as in the hybrid setup, we are able to directly compare results from the simulated and real experiments.

Fig. 24 shows the outcomes of different grasping methods for individual objects. It can be seen that for the weakly textured kitchen container, the contour method is the only method that produces stable grasps. Also for the weakly textured red basket, the contour grasps perform best, with over 80% success when both stable and less stable grasps are taken into account. Both in the case of the kitchen container and the red basket, a big amount of collisions occurred, due to the fact that the object surfaces are not detected.

For textured objects, the success of different methods varies across objects, which once more confirms the supplementary strength of different methods, as seen in the previous results from Sect. 7.1 and Sect. 7.2. The difference compared to the results from the hybrid setup is that the three-finger $sb_3\mathrm{EGA}_1$ does not perform as well. The $sb_3\mathrm{EGA}_1$ is an encompassing grasp of the whole surface and this fits well with the objects used in the simulated setup, objects that were mostly closed, with well-textured top surface. Two such objects from the real experiments are corny and marmalade, and the increase in the performance of the $sb_3\mathrm{EGA}_1$ method for the two objects can be seen in Fig. 24.

In Fig. 25(top), the outcomes of different grasping methods are averaged over all objects. The success rate of the first grasping attempt varies between 30-60%, in agreement with results from the hybrid setup.

As can be expected, the results for the objects in the upright position are better than for the objects laying down. The results for the two object scenes are better when the objects are far apart, see Fig. 25(bottom).

One of the differences between the real world and simulated three-finger setups is that in the case of simulations, all suggested grasps are reachable, or in other words the simulated hand can be placed in all positions with respect to the object. In the case of the real setups, only the grasps that can be reached with the robot arm are feasible. In a setup where the stereo camera and the robot arm are placed close together, analog to the configuration of human body, the visible sides of objects will also be the reachable sides of the objects. This means that the grasps involving the occluded side of the object, that are more likely to fail because of lack of information, are naturally filtered out. Therefore the results show fewer number of collisions in the real experiments Fig. 24 compared to Fig.21.

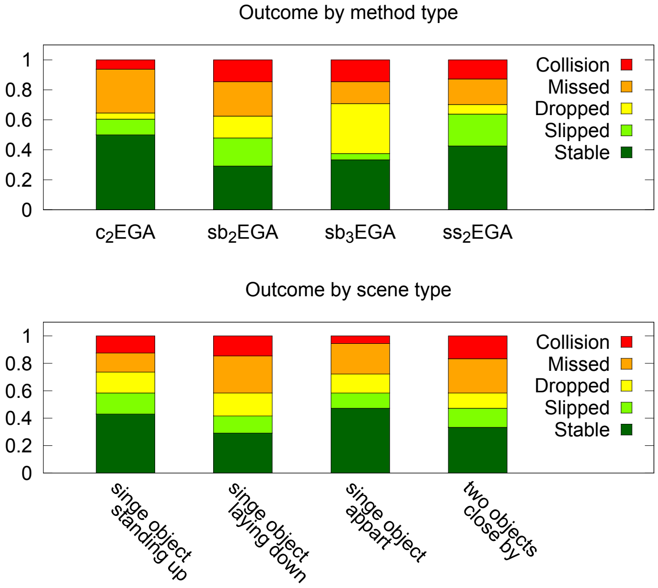


Figure 25: Grasp results for the real-world setup – three-finger dexterous hand experiments. The top figure shows the distribution of outcomes of different grasping methods for all experiments. The bottom figure shows the distribution of outcomes for different types of scenes.

8 Discussion

In this paper, we presented a bottom-up vision system for general scene interpretation in terms of hierarchies of visual descriptors, and we used this system for grasping unknown objects. We continued our earlier work (Popović et al., 2010), by extending the hierarchical representation of the Early Cognitive Vision (ECV) system to the texture and surface domain, and by using the representation to generate not only contour-based, but also surface-based grasps.

The ECV system organizes multi-modal three-dimensional visual information in a hierarchy of increasing level of abstraction. In the contour domain, local edge features on the lowest level are grouped into contours, and contours belonging to the same surface are further grouped together. In the texture domain, local textured patches on the lowest level are grouped into larger surflings, which are then grouped to form surfaces. This approach has three advantages: 1) it allows us to address the grasping problem at a level in the hierarchy which sufficiently narrows the search space for grasping possibilities, 2) the vision system extracts rich visual information about surfaces of objects in the scene, which allows to extract grasp-

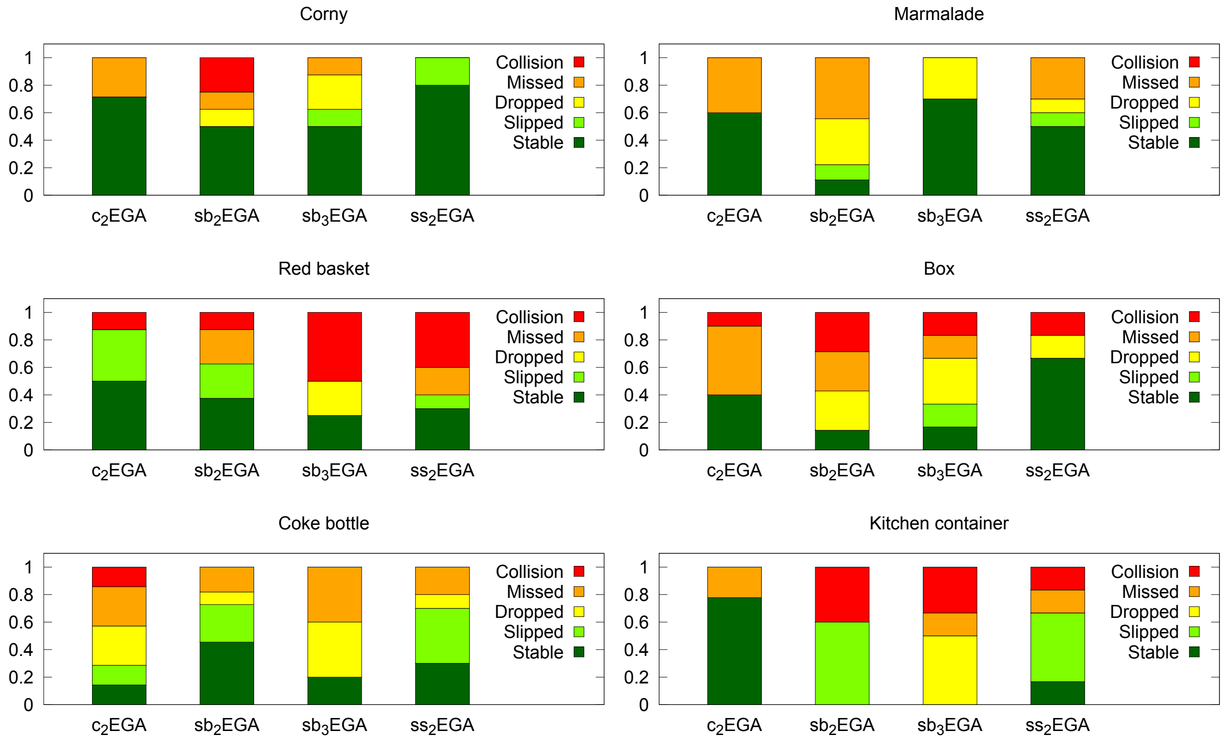


Figure 24: Grasp results for the real-world setup – three-finger dexterous hand experiments. The figures show the distribution of outcomes of different grasping methods for individual objects, and are averaged over different scene types. On average eight grasping attempts were made per method for each object.

ing hypotheses, and 3) by using lower levels of hierarchy we can define good contact points. We proposed different *elementary grasping actions*, utilizing contour and surface information present in the hierarchical representation made of the scene.

We tested the system in different experimental setups. First, we used a hybrid real-world and simulated setup. Based on real stereo images, our method built a visual representation of the scene and generated grasps. These grasps were then executed in a dynamic simulator. This setup allowed us to test a large number of grasps and get quantitative results, while still dealing with the noise and uncertainty in the real-world visual data. It allowed us, moreover, to compare the different methods under the exact same circumstances. Second, to show the applicability of the proposed methods on real robotic systems, we tested the proposed methods on two different real-world experimental setups, using a parallel and a three-finger dexterous hand.

The results show a good average performance of the proposed grasping methods, which is even further improved when the different methods are combined. This shows that the different methods, which are based on different types of visual information and apply different grasps to contact points, complement each other so that a better performance can be achieved. In particular, the contour-based methods perform better in the situations where objects are not textured, while different surface-based methods complement each other in the situations where objects are textured. By combining both types of information, the overall system can deal with both types of objects.

We proposed two surface-based methods. The SVD method uses a rectangular approximation of the surface, and therefore suggests a limited number of grasps. The boundary method suggests a larger number of grasps, based on a more general representation of the surface, which can deal with a larger variety of shapes.

If we compare the overall results of the single-object scenes with the complex scenes, we see that the success rates are in the same range. This shows that our hierarchical vision system provides a powerful representation of the scene that can be used to generate good grasps, even with increasing visual complexity. Similarly, in the hybrid setup, the results for the scenes with textured and non-textured background are similar, which also indicates that the proposed methods are robust to different levels of visual complexity.

An issue with the proposed method is that we cannot predict collisions of the gripper with the back side of objects, because of the lack of features due to occlusions. To deal with this, we propose to extend the ECV to not only rely on bottom-up processes, but to also let the system make top-down projections about the shape of the object, in order to hallucinate the back sides of objects.

The experiments from the hybrid setup provided an in-depth analysis of the grasping methods and their sub-types. The experiments from the two real-world setups confirmed that the same level of success can be expected when grasping in real-world scenarios. Performing the experiments on the different setups and with different objects has helped to better understand the proposed grasping methods.

Since we are dealing with grasping unknown objects in unknown scenes, a 100% success is not to be expected. However, the results show that a good average performance can be achieved and that the grasp success strongly increases when more attempts are made and different grasp types are combined. In the current system, the different grasp hypotheses for a scene are not ordered, but a hypothesis is selected at random from the possibilities. Using developmental learning mechanisms, such as proposed in (Kraft et al., 2008), a system can learn to improve grasping success further based on acquired grasp experience allowing to order the grasp hypotheses for more efficient grasp selection.

The proposed grasping method can as well be used for grasping known objects, provided that the pose of the object can be estimated. In the case of dealing with known objects, the increased performance can be expected. Due to the relatively small number of grasps suggested by our system, an on-line simulation can be used to select good grasps by testing all proposed grasps. When provided with the complete model of an object, the collisions can be detected also with the parts of the object that are not visible from the given stereo view.

The VisGraB benchmark that we used in this paper is open for scientific use. The stereo images, the simulated environment, and the dynamic simulator are available on http://www.csc.kth.se/visgrab.

Acknowledgements

This work was supported by the EU through the projects XPERIENCE (FP7-ICT-270273), eSMCs (FP7-IST-270212), and by the Swedish Foundation for Strategic Research.

References

E. Başeski, N. Pugeault, S. Kalkan, L. Bodenhagen, J. H. Piater, and N. Krüger. Using Multi-Modal 3D Contours and Their Relations for Vision and Robotics. *Journal of Visual Communication and Image Representation*, 21(8):850–864, 2010.

L. Bodenhagen, D. Kraft, M. Popović, E. Baśeski, P. E. Hotz, and N. Krüger. Learning to grasp unknown objects based on 3d edge information. In *Proceedings of the 8th IEEE international conference on Computational intelligence in robotics and automation*, 2009.

J. Bohg and D. Kragic. Learning grasping points with shape context. *Robotics and Autonomous Systems*, 58(4):362–377, 2010.

C. Borst, M. Fischer, and G. Hirzinger. Grasping the dice by dicing the grasp. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3692–3697, 2003.

E. Chinellato, A. Morales, R. B. Fisher, and A. P. del Pobil. Visual quality measures for characterizing planar robot grasps. *IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews*, 35(1):30–41, 2005.

M. T. Ciocarlie and P. K. Allen. Hand posture subspaces for dexterous robotic grasping. *The International Journal of Robotics Research*, 28(7):851–867, 2009.

N. Curtis and J. Xiao. Efficient and effective grasping of novel objects through learning and adapting a knowledge base. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008.

R. Detry, N. Pugeault, and J. Piater. A probabilistic framework for 3D visual object representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(10): 1790–1803, 2009.

R. Detry, D. Kraft, O. Kroemer, L. Bodenhagen, J. Peters, N. Krüger, and J. Piater. Learning grasp affordance densities. *Paladyn Journal of Behavioral Robotics*, (accepted), 2011.

C. Dune, E. Marchand, C. Collowet, and C. Leroux. Active rough shape estimation of unknown objects. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008.

S. El-Khoury and A. Sahbani. Handling objects by their handles. In *Proceedings of IROS 2008 Workshop on Grasp and Task Learning by Imitation*, 2008.

L. Ellekilde and J. Jørgensen. Usage and verification of grasp simulation for industrial automation. In *Proceedings of 42st International Symposium on Robotics (ISR)*, Chicago, 2011.

L. F. Gallardo and V. Kyrki. Detection of parametrized 3-d primitives from stereo for robotic grasping. In *Advanced Robotics (ICAR), 2011 15th International Conference on*, pages 55 –60, june 2011.

C. Goldfeder, P. K. Allen, C. Lackner, and R. Pelossof. Grasp planning via decomposition trees. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'07)*, 2007.

K. Hübner, S. Ruthotto, and D. Kragic. Minimum volume bounding box decomposition for shape approximation in robot grasping. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'08)*, pages 1628–1633, 2008.

J. Jørgensen, L. Ellekilde, and H. Petersen. Robworksim - an open simulator for sensor based grasping. In *Proceedings of Joint 41st International Symposium on Robotics (ISR 2010) and the 6th German Conference on Robotics (ROBOTIK 2010)*, Munich, 2010.

A. Kjær-Nielsen, A. G. Buch, A. E. K. Jensen, B. Møller, D. Kraft, N. Krüger, H. G. Petersen, and L.-P. Ellekilde. Ring on the hook: Placing a ring on a moving and pendulating hook based on visual input. 28(3):301 – 314, 2010.

G. Kootstra, M. Popović, J. Jørgensen, D. Kragic, H. Petersen, and N. Krüger. Visgrab: A benchmark for vision-based grasping. *Paladyn Journal of Behavioral Robotics*, submitted.

D. Kraft, N. Pugeault, E. Başeski, M. Popović, D. Kragic, S. Kalkan, F. Wörgötter, and N. Krüger. Birth of the object: Detection of objectness and extraction of object shape through object-action complexes. *International Journal of Humanoid Robotics*, 5(2):247–265, 2008.

N. Krüger et al. Early cognitive vision as a front-end for cognitive systems. In *ECCV 2010 Workshop on "Vision for Cognitive Tasks"*, 2010.

A. Miller and P. Allen. Graspit! a versatile simulator for robotic grasping. *Robotics Automation Magazine, IEEE*, 11 (4):110 – 122, dec. 2004. ISSN 1070-9932.

A. T. Miller, S. Knoop, H. I. Christensen, and P. K. Allen. Automatic grasp planning using shape primitives. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'03)*, pages 1824–1829, 2003.

A. Morales. *Learning to predict grasp reliability with a multifinger robot hand by using visual features*. PhD thesis, Univertitat Jaume I, Castellón, Spain, 2004.

A. Morales, P. J. Sanz, A. P. del Pobil, and A. H. Fagg. Vision-based three-finger grasp synthesis constrained by hand geometry. *Robotics and Autonomous Systems*, 54(6):496 – 512, 2006. ISSN 0921-8890.

W. Mustafa, M. Popović, J. Jessen, D. Kraft, S. M. Olesen, A. G. Buch, and N. Krüger. Using surfaces and surface relations in an early cognitive vision system. 2011.

V. Nguyen. Constructing stable grasps. *International Journal on Robotic Research*, 8(1):26–37, 1989.

R. Pelossof, A. Miller, P. Allen, and T. Jebara. An svm learning approach to robotic grasping. In *Proceedings of the IEEE Conference on Robotics and Automation*, 2004.

J. Ponce and B. Faverjon. On computing three-finger force-closure grasps of polygonal objects. *Robotics and Automation, IEEE Transactions on*, 11(6):868 –881, Dec. 1995. ISSN 1042-296X. doi: 10.1109/70.478433.

M. Popović, D. Kraft, L. Bodenhagen, E. Başeski, N. Pugeault, D. Kragic, T. Asfour, and N. Krüger. A strategy for grasping unknown objects based on co-planarity and colour information. *Robotics and Autonomous Systems*, 58(5):551 – 565, 2010. ISSN 0921-8890. doi: DOI:10.1016/j.robot.2010.01.003.

M. Popović, G. Kootstra, J. Jørgensen, D. Kragic, and N. Krüger. Grasping unknown objects using an early cognitive vision system for general scene understanding. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS), September 25-30, San Francisco, CA.*, 2011.

N. Pugeault, F. Wörgötter, and N. Krüger. Visual primitives: Local, condensed, and semantically rich visual descriptors and their applications in robotics. *International Journal of Humanoid Robotics (Special Issue on Cognitive Humanoid Vision)*, 7(3):379–405, 2010a.

N. Pugeault, F. Wörgötter, and N. Krüger. Disambiguating multimodal scene representations using perceptual grouping constraints. *PLoS ONE*, 5(6):e10663, 06 2010b.

D. Rao, Q. V. Le, T. Phoka, M. Quigley, A. Sudsang, and A. Y. Ng. Grasping novel objects with depth segmentation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

M. Richtsfeld and 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*, 2008.

A. Saxena, J. Driemeyer, and A. Y. Ng. Robotic grasping of novel objects using vision. *The International Journal of Robotics Research*, 27(2):157–173, 2008.

K. Shimoga. Robot grasp synthesis algorithms: A survey. *International Journal on Robotic Research*, 15(3):230–266, 1996.

Learning Continuous Grasp Stability for a Humanoid Robot Hand Based on Tactile Sensing

J. Schill and J. Laaksonen and M. Przybylski and V. Kyrki and T. Asfour and R. Dillmann

Abstract— **Grasp stability estimation with complex robots in environments with uncertainty is a major research challenge. Analytical measures such as force closure based grasp quality metrics are often impractical because tactile sensors are unable to measure contacts accurately enough especially in soft contact cases. Recently, an alternative approach of learning the stability based on examples has been proposed. Current approaches of stability learning analyze the tactile sensor readings only at the end of the grasp attempt, which makes them somewhat time consuming, because the grasp can be stable already earlier.**

In this paper, we propose an approach for grasp stability learning, which estimates the stability continuously during the grasp attempt. The approach is based on temporal filtering of a support vector machine classifier output. Experimental evaluation is performed on an anthropomorphic ARMAR-IIIb. The results demonstrate that the continuous estimation provides equal performance to the earlier approaches while reducing the time to reach a stable grasp significantly. Moreover, the results demonstrate for the first time that the learning based stability estimation can be used with a flexible, pneumatically actuated hand, in contrast to the rigid hands used in earlier works.

I. INTRODUCTION

The sense of touch is essential to human grasping. The work described in this paper considers robotic tactile sense as a biomimetic replacement for the sense of touch, especially when estimating grasp stability. Grasp stability in analytical sense is well defined and can be readily computed in simulation where enough data of the grasp is available, i.e. all contacts between the robotic hand and the object that is grasped. Additionally, using a force closure metric for grasp stability, one can compute a grasp that sufficiently resists outside forces, such as gravity, thus allowing the robot to manipulate the object, for example by lifting the object. However, when using real hardware, the tactile sensor data is imperfect, both in the sense of detecting contacts and in the sense of determining the actual contact forces. In some cases the proprioceptive information, i.e. joint configuration, is also difficult to determine accurately, thus, causing uncertainty in ascertaining the kinematic configuration of the hand. All these described phenomena pave a difficult road for computing the grasp stability analytically with real hands.

In this paper, we focus on learning the grasp stability instead of analytically solving it. Compared to the analytical methods, learning requires training data, which needs to be collected beforehand. As the training data, we can use any pertinent data that can be collected from robotic hand, in our case we use input from all tactile sensors and the hand finger configuration. It is also important to notice that the raw sensor data can be used in the learning, for example, there is no need to know the kinematic configuration of the hand to compute the true locations of the contacts when analytically solving the grasp stability. This feature allows grasp stability to be learned for many different robotic hands with only minimum changes.

There has been a number of publications on learning the grasp stability [1], [2]. These approaches evaluate the stability after the hand finished closing around an object. We extend the work presented in previous papers, so that the decision on the grasp stability can be achieved during the grasping instead of at the end of the grasp. We also demonstrate that the learning of the grasp stability is possible with the ARMAR-IIIb hand [3], [4] , a flexible anthropomorphic hand operating on pneumatics.

The rest of the paper is divided into four sections. Section II gives an overview on learning grasp stability as well as other learning approaches that are grasp and manipulation related. Section III introduces a theoretical background on machine learning methods and how they can be applied to the grasp stability problem. Section IV contains the experiments made on data collected using the ARMAR-IIIb hand. We conclude with discussion of the results in Section V.

II. RELATED WORK

Grasp stability analysis by analytical means is a well established field. However, to analytically determine the grasp stability, the kinematic configuration of the hand and the contacts between the hand and the object must be perfectly known. Previous studies on this subject are numerous and [5] gives a detailed review. However, the references are useful only in cases when conditions described above are true. When this is the case, it is possible to determine if the grasp is either force or form closure grasp [6], which ensures the stability. Compared to this body of work, we wish to learn the stability from existing data, i.e. the tactile data.

The research on use of tactile and other sensors in a grasping context has increased in last few years. Felip and Morales [7] developed a robust grasp primitive, which tries to find a suitable grasp for an unknown object after a few initial

The research leading to these results has received funding from the European Community's Seventh Framework Programme GRASP under grant agreement n° 215821 and Xperience under grant agreement n° 270273

J. Laaksonen and V. Kyrki are with Department of Information Technology, Lappeenranta University of Technology, P.O. Box 20, 53851 Lappeenranta, Finland, jalaakso@lut.fi, kyrki@lut.fi

M. Przybylski, J. Schill, T. Asfour and R. Dillmann are with the Humanoids and Intelligence Systems Lab, Karlsruhe Institute of Technology, Karlsruhe, Germany, {markus.przybylski, schill, asfour, dillmann}@kit.edu

grasp attempts. However, only finger force sensors were used in the study.

Apart from using tactile information as a feedback for low level control [8], tactile sensors can be used to detect or identify object properties. Jiméneza et al. [9] use the tactile sensor feedback to determine what kind of a surface the object has, which is then used to determine a suitable grasp for an object. Petrovskaya et al. [10] on the other hand use tactile information to reduce the uncertainty of the object pose, upon an initial contact with the object. In their work, a particle filter is used to estimate object's pose, but the tactile sensor used to detect contact with the object is not embedded in the gripper performing the grasping.

Object identification has been studied by Schneider et al. [11] and Schöpfer et al. [12]. Schneider et al. show that it is possible to identify an object using tactile sensors on a parallel jaw gripper. The approach is similar to object recognition from images and the object must be grasped several times before accurate recognition is achieved. Schöpfer et al. use a tactile sensor pad fixed to a probe instead of a gripped or hand. They also study on different temporal features which can be used to recognize objects. Similar object recognition systems have been presented in [13], [14].

The approach used and extended here has been published in [1]. Similar approach was used in [2]. However, in this paper we show that we can use described methods with a more complex hand, the ARMAR-IIIb humanoid hand, and that we can extend the single time instance classifier by means of filtering.

III. Supervised Learning of Grasp Stability

A. Learning Grasp Stability

Our notation of observations follows [1]:

- $D = [o_i], i = 1, \ldots, N$ denotes a data set with N observation sequences.
- $o_i = [x^i_t], t = 1, \ldots, T_i$ is an observation sequence with T_i samples.
- $x^i_t = [\mathbf{f}^i_t \ \mathbf{j}^i_t]$, each sample consists of $\mathbf{f}$, the features extracted from tactile sensors and $\mathbf{j}$, the joint configuration.

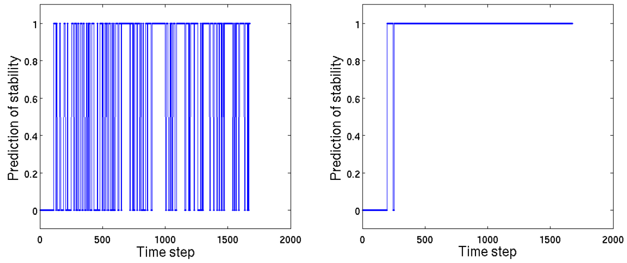
To learn grasp stability, the training data is collected from series of grasps, noted by the observation sequences o_i. Each observation sequence is labeled with a label indicating either a stable or unstable grasp. Then, from each observation sequence the last sample, $x^i_{T_i}$, is used for the training. This captures the time instant on which the decision of stable or unstable grasp is based on. Both unstable and stable grasp must be included in the training data so that sufficient data is available to discern the stable grasps from the unstable grasps.

We use Support Vector Machine (SVM) [15] to classify the grasp as either stable or unstable. Compared to force closure metric from the analytical methods for computing the grasp stability, the binary classification is not as informative as the continuous value given by the force closure metric, however the classification result reflects the stability criteria in the training data directly. Another benefit of SVM is that it is computationally efficient, so that it can be used on-line during grasping.

B. Learning Temporal Changes in Grasp Stability

In [1], the temporal information collected during a grasp is used in conjunction with a hidden Markov model (HMM) to decide whether the grasp is stable or not. But for the method to be able to decide, the grasp must be completed. The second method presented in [1] was based on the idea depicted in III-A. We propose to extend the instantaneous SVM-based method by applying the learned stability model on-line to each sample $x_1, \ldots, x_T$ we obtain during the grasp, contrary to the previous approach, where only the final sample, x_T, is is used to determine the stability of the grasp. This extension allows quicker decision making on the grasp quality in the case of a stable grasp.

As the method described in III-A does not remember any of the previous time instances and does not consider the whole grasp sequence from $t = 1, \ldots, T$, the classification result over time may oscillate. One pathological example is shown Figure 1. Through the use of filtering and thresholding, the oscillations can be effectively removed.



(a)

(b)

Fig. 1: (a): Each time instance of a stable grasp classified with a SVM classifier; (b): The classification result filtered with an exponential filter and thresholded.

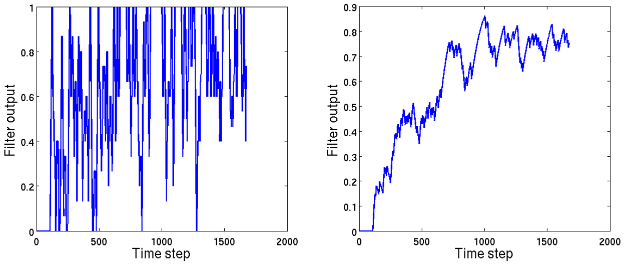
We study two different filter types: a mean filter and an exponential filter. The results of the experiments with the filters are shown in Section IV. The input for the filters are the results from the classifier, either 0 or 1. The mean filter can be defined as a sliding window, with window size w. The mean of the data in the window is then calculated, and this result is the output of the filter. Exponential filter is described by

$$y(t) \ = \ (1 - \alpha) \cdot y(t - 1) + \alpha \cdot x(t) \ . \qquad (1)$$

Equation 1 consists of $y(t)$ and $y(t-1)$, filter output at time instances t and $t-1$, of $x(t)$ the binary stability at time t and of α which a weighting factor. An examples of both filters are shown in Figure 2 which depicts the same sequence as in Figure 1.

Introducing the filters requires setting more parameters in addition to the parameters for SVM. These include w for the mean filter window width, and α for the exponential filter. In addition both require the threshold, thr, for the binary decision of stability. After the threshold has been crossed, the grasp is deemed stable. Close to optimal parameters can be found experiementally and we have done that for the datasets used in this paper.

In addition to the filters, we ran experiments without using any filters, thus, the output from the classifier is taken directly. This approach provides a quicker response to stable grasps but can also misclassify unstable grasps as stable grasps more frequently than the filter based approach.



(a)

(b)

Fig. 2: (a): Filter output of mean filter; (b): Filter output of exponential filter.

C. Feature Extraction

Each of the tactile sensors on the ARMAR-IIIb platform produces a tactile image. An example image showing all six tactile images is shown in Figure 3. This imaging property of the sensors allows us to use image feature extraction techniques. In this case we have chosen the image moments as our feature extractor, which have been shown to perform well in this task [16]. The hand comprises of two different sizes of tactile sensors which contain 4x7 or 4x6 tactile elements or taxels.

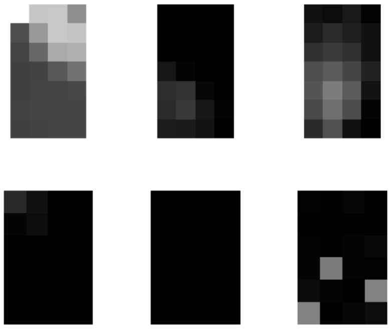


Fig. 3: Tactile images from ARMAR-IIIb.

Raw image moments are defined as

$$m_{p,q} = \sum_x \sum_y x^p y^q I(x,y) \ , \qquad (2)$$

where $I(x,y)$ is the force measured by the taxel. The moments are computed up to order two, that is $(p+q) = o, \ \ o = \{0,1,2\}$. These are related to the total pressure, the mean of the contact area, and the shape of the contact area, indicated by the variance in x- and y-axes. Moments are computed for all tactile sensors individually, thus $\mathbf{f} \in \mathbb{R}^{36}$.

In addition to the tactile images, the joint angle sensors provide a source of information relevant to the stability of the grasp. However as the number of fingers and joints is usually much less than the number taxels (tactile sensing elements) in tactile sensors, it is reasonable to use the data from the joints directly. In this case, 8 joint angle sensors are available, thus $\mathbf{j} \in \mathbb{R}^8$. All feature vectors, x_t^i, were normalized to zero mean and unit standard deviation.

IV. EXPERIMENTS

A. Hardware Platform

We used the humanoid robot ARMAR-IIIb as a test platform for the experiments with our stability classifier. ARMAR-IIIb consists of several kinematic subsystems: The head, the torso, two arms, two hands, and the platform. The head has seven degrees of freedom (DoF) and contains four cameras, i.e. two cameras per eye. The torso has 1 DoF in the hip, allowing the robot to turn its upper body. Each of the two 7 DoF arms consists of a 3 DoF shoulder, a 2 DoF elbow and a 2 DoF wrist. At the tool center point (TCP) of each arm a FRH-4 Hand [17] is mounted. The hands are pneumatically actuated using fluidic actuators. For the experiments in this paper, we used ARMAR-IIIb's right hand, (see Fig. 4), which is equipped with joint encoders and pressure sensors. This allows a force position control of each DoF [18]. The hand has 1 DoF in the palm, and 2 DoF in the thumb, the index and the middle finger, respectively. Apart from that, there is 1 DoF for combined flexion of the pinky and ring finger. Furthermore the hand contains 6 tactile sensors from Weiss Robotics [19]. One tactile sensor is mounted on the distal phalanges of the thumb, the index and the middle finger, respectively. Three tactile sensors are mounted at the palm, in the area between the thumb and the index and middle fingers. The tactile sensors have a resolution of 4 × 7 taxel (phalanges) and 4 × 6 taxel (palm). They use a resisitive working principle to measure the pressure applied to the sensor. Therefore an array of electrodes is covered with a layer of conductive foam. When a pressure is applied to the sensor the resistance between the electrodes decreases, which is measured by an microcontroller. Further information can be found in [20], [21], [22].

B. Data Collection

In order to provide sensor input for the training and the validation of the classifier, we needed to treat two distinct cases:

- Collect data for successful, stable grasps.
- Collect data for unstable grasps.

The second case also includes data for the cases where the hand cannot close completely or not at all,due to obstacles, and cases where the hand closes emptily, i.e. it does not

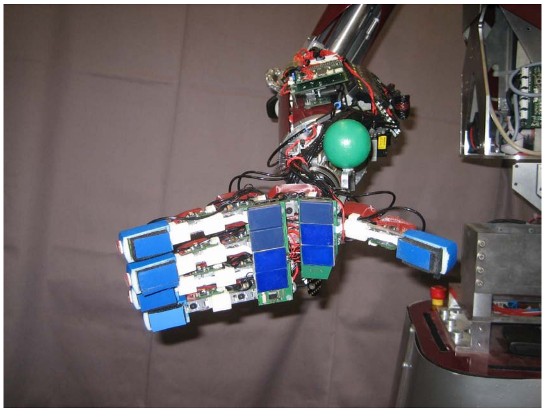


Fig. 4: ARMAR's right hand. Tactile sensors are mounted on the palm and the distal phalanges of the thumb, the index and the middle finger.

experience contact to any object at all. Yet in all these cases one gets sensor readings that have to be considered for training and validating the classifier. We collected data from the following two types of sensors:

- Tactile sensor data
- Joint angle data of the hand joints

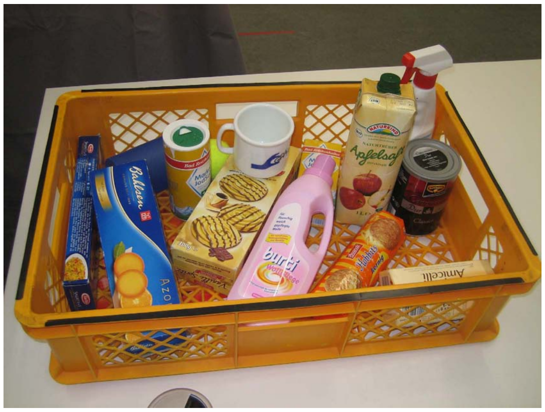


Fig. 5: The basket with our test objects.

For data collection, we executed grasps on a set of household items located in a box (see Fig. 5). The configuration of the objects in the box was modified between the individual test runs in order to allow the hand to reach a large variety of different end configurations. We used the following data collection procecure: First, we placed the box with the objects in front of the robot. Then we moved ARMAR's right hand to a pre-grasp pose near the target object. Different possible pre-grasp poses included the following:

- Grasps from the top where the hand would move vertically down.
- Grasps from the top, but with tilted approach directions.
- Grasps from the side.
- Varying roll angles of the hand with respect to the approach direction, for each of the three cases above.

After moving the hand to the pre-grasp pose, we started the data recording which means we began to read and store the tactile sensor data and joint angle data once during every pass of ARMAR's control loop. All data were labeled with a time stamp. In the next step, we moved the hand towards the object until the tactile sensors in the palm reported contact with the object. Then we closed the hand and waited until the pressure on the hand's actuator stabilized and would not grow anymore. The finger forces are set to the maximum to create a strong tactile image on the sensors. Due to the compliant characteristic of the hand, the hand adapts to the shape of the object. In this context we point out that we considered only three-fingered grasps, i.e. we only closed the thumb, the index and the middle finger during grasping. We did not close the ring and small finger, as they are not equipped with tactile sensors and thus they would not contribute to the tactile sensor input of the classifier. After closing the hand, we stopped the recording of the sensor data. Finally, we tried to lift the object by moving ARMAR's hand up. Then, we reported the result of the experiment, i.e. whether the grasp was successful or not. We repeated the above procedure until enough samples had been collected. We collected two separate sets, D_1 and D_2. D_1 contained 71 stable grasps and 94 unstable grasps. D_2 comprised of 82 stable grasps and 76 unstable grasps. By collecting two separate sets with different grasps, we can get an idea of the generalization capability of the classifier which was tested in the validation tests. Figures 6 and 7 show some successful grasps from the validation tests. The left column shows the situation after closing the hand. The right column shows the grasps after lifting the respective object.

C. Experimental Results

We have divided the experiments into two parts. The first part consists of synthetic tests, which presents the reliability and accuracy of the classification of the grasp stability and comparisons between different filter types. The second part is validation test, using a learned stability model with the real ARMAR-IIIb platform.

1) Synthetic tests: In the synthetic tests, we used both datasets D_1 and D_2. For most experiments, confusion matrix is presented, showing how the classifier performs in terms of true positives (stable, predicted stable), false positives (unstable, p. stab.), true negatives (unstable, p. unstab.) and false negatives (stable, p. unstab.).

In Table I, the SVM was trained with data from corresponding dataset, only the last sample from each observation sequence was classified, to enable comparison to earlier works. The reported results are averages from 10-fold cross validation. The results show that the performance across datasets is similar. These results can be compared with reported results in [1], [2], showing that the ARMAR-IIIb hardware is able to reach similar performance as the Schunk Dextrous Hand (SDH) or the Barrett hand in this task.

Contrary to results in Table I, in Tables II, III and IV the whole observation sequence was classified using the methodology presented in Section III-B. In Table II, the mean filter was used with window width of 25 and with threshold of 0.61, Table III shows result with an exponential filter with

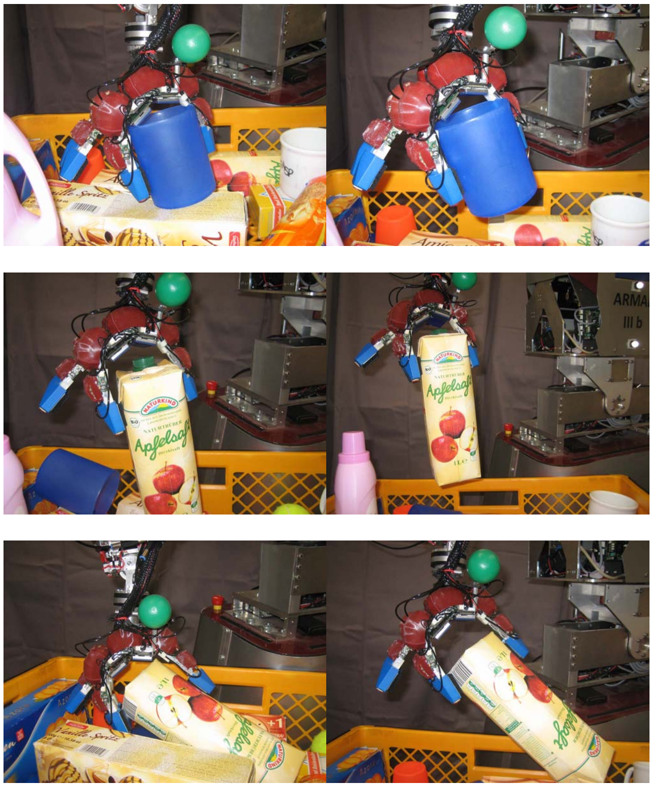


Fig. 6: Some example grasps. Left column: situation immediately after closing the hand. Right column: After lifting the object.

TABLE I: Confusion matrix for classification rates of grasps when classifying only the last sample, for datasets D_1 and D_2.

D_1	P. Stab.	P. Unstab.	D_2	P. Stab.	P. Unstab.
Stable	0.79	0.21	Stable	0.72	0.28
Unstable	0.28	0.72	Unstable	0.26	0.74

$\alpha = 0.056$ and threshold of 0.61. These parameter values were searched for using grid search and produced the best results for both datasets. Results in Table IV were obtained without using a filter.

Overall, when using a filter with the classification, the overall classification rate is similar to the last sample classification, but classification rate of the unstable grasps is better. This can be explained through the use of the filter which filters out the effect of the last sample, thus, leading to a better classification result. In the case where no filters are used, in Table IV, the stable grasps are predicted well, but this translates also to falsely predicting that unstable grasps are stable. On average, the filter based classification is better in predicting the stable and unstable grasps across the two datasets.

One interesting possibility that comes with the method described in Section III-B is that the grasp sequence can be stopped when the classifier decides that a stable grasp

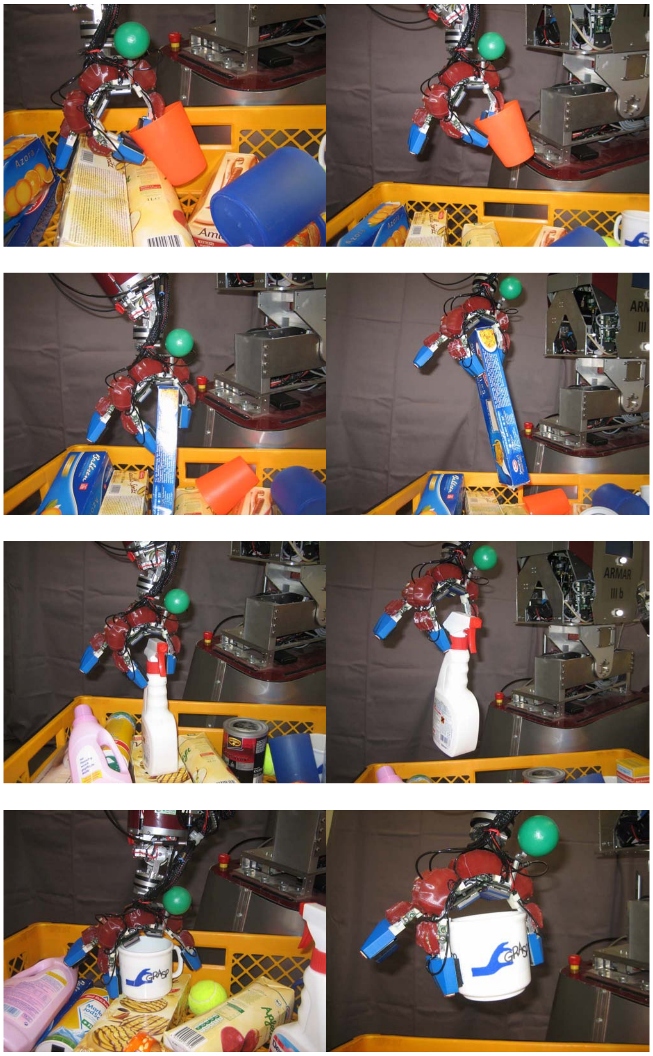


Fig. 7: Some example grasps. Left column: situation immediately after closing the hand. Right column: After lifting the object.

TABLE II: Confusion matrices for classification rates of grasps using mean filter ($w = 25$, thr $= 0.61$).

D_1	P. Stab.	P. Unstab.	D_2	P. Stab.	P. Unstab.
Stable	0.77	0.23	Stable	0.74	0.26
Unstable	0.24	0.76	Unstable	0.16	0.84

TABLE III: Confusion matrices for classification rates of grasps using exponential filter ($\alpha = 0.056$, thr $= 0.61$).

D_1	P. Stab.	P. Unstab.	D_2	P. Stab.	P. Unstab.
Stable	0.79	0.21	Stable	0.73	0.27
Unstable	0.23	0.77	Unstable	0.16	0.84

TABLE IV: Confusion matrices for classification rates of grasps without a filter.

D_1	P. Stab.	P. Unstab.	D_2	P. Stab.	P. Unstab.
Stable	0.90	0.10	Stable	0.87	0.13
Unstable	0.46	0.54	Unstable	0.21	0.79

TABLE V: Confusion matrices for validation tests.

Mean filt.	P. Stable	P. Unstable
Stable	0.77	0.23
Unstable	0.39	0.61
Exp. filt.	P. Stable	P. Unstable
Stable	0.76	0.24
Unstable	0.38	0.62
No filter	P. Stable	P. Unstable
Stable	0.90	0.10
Unstable	0.46	0.54

has been achieved. Using a mean filter, the decision time was 68.6 % of the whole grasp sequence on average, with a exponential filter, the time was 66.9 % and without a filter the time was 59.6 % For example, if a whole grasp sequence is 1000 time steps long, the classification using a mean filter can stop the grasp at time step 686 on average, if the grasp is a stable grasp. Without a filter, the average time goes down as expected but with a cost of overall classification rate as seen in Table IV.

2) Validation tests: To mimic a real world usage scenario, dataset D_1 was used to train the SVM classifier. Then using the trained classifier, dataset D_2 was classified. Each observation sequence in the dataset was classified with mean and exponential filters and without filtering. The results are show in Table V. Compared to results in Table I, the number of false positives rises. This effect might be due to tactile sensor hysteresis, i.e. the output from the sensors changes between the collection of datasets which in turn means that dataset D_1 does not represent the data in D_2 and leads to worse results.

V. CONCLUSIONS

In this paper, we focused on learning grasp stability from labeled data, similar to approaches in [1], [2]. We utilized a well-known classifier, SVM, and trained it using grasp data acquired from the sensors of the humanoid hand of ARMAR-IIIb. We showed that we are able to reach similar results with ARMAR-IIIb as previously reported on other types of hardware, such as Schunk Dextrous Hand or Barrett hand. We also extended the SVM based grasp stability classifier with use of filters to whole grasp sequence instead of just the end of the grasp sequence. This allows faster decisions for stable grasps.

REFERENCES

[1] Y. Bekiroglu, J. Laaksonen, J. A. Jørgensen, V. Kyrki, and D. Kragic, “Assessing grasp stability based on learning and haptic data,” *Robotics, IEEE Transactions on*, vol. 27, no. 3, pp. 616 –629, 2011.

[2] H. Dang, J. Weisz, and P. K. Allen, “Blind grasping: Stable robotic grasping using tactile feedback and hand kinematics,” in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, may 2011, pp. 5917 –5922.

[3] T. Asfour, K. Regenstein, P. Azad, J. Schröder, N. Vahrenkamp, and R. Dillmann, “ARMAR-III: An Integrated Humanoid Platform for Sensory-Motor Control,” in *IEEE-RAS International Conference on Humanoid Robots*, Genova, Italy, December 2006, pp. 169–175.

[4] T. Asfour, P. Azad, N. Vahrenkamp, K. Regenstein, A. Bierbaum, K. Welke, J. Schröder, and R. Dillmann, “Toward Humanoid Manipulation in Human-Centred Environments,” *Robotics and Autonomous Systems*, vol. 56, pp. 54–65, January 2008.

[5] A. Bicchi and V. Kumar, “Robotic grasping and contact: A review,” in *ICRA*, 2000, pp. 707–714.

[6] D. Prattichizzo and J. C. Trinkle, “Grasping,” in *Springer Handbook of Robotics*, 1st ed., B. Siciliano and O. Khatib, Eds. Berlin, Germany: Springer-Verlag, 2008.

[7] J. Felip and A. Morales, “Robust sensor-based grasp primitive for a three-finger robot hand,” in *IEEE/RSJ International. Conference on Intelligent Robots and Systems*, Oct. 2009.

[8] T. Tsuboi and et al., “Adaptive grasping by multi fingered hand with tactile sensor based on robust force and position control,” in *IEEE International Conference on Robotics and Automation*, 2008, pp. 264–271.

[9] A. Jiménez, A. Soembagijo, D. Reynaerts, H. V. Brussel, R. Ceres, and J. Pons, “Featureless classification of tactile contacts in a gripper using neural networks,” *Sensors and Actuators A: Physical*, vol. 62, no. 1-3, pp. 488–491, 1997.

[10] A. Petrovskaya, O. Khatib, S. Thrun, and A. Y. Ng, “Bayesian estimation for autonomous object manipulation based on tactile sensors,” in *ICRA*, 2006, pp. 707–714.

[11] A. Schneider, J. Sturm, C. Stachniss, M. Reisert, H. Burkhardt, and W. Burgard, “Object identification with tactile sensors using bag-of-features,” in *In Proc. of the International Conference on Intelligent Robot Systems (IROS)*, 2009.

[12] M. Schöpfer, M. Pardowitz, and H. J. Ritter, “Using entropy for dimension reduction of tactile data,” in *14th International Conference on Advanced Robotics*, ser. Proceedings of the ICAR 2009, IEEE. Munich, Germany: IEEE, 22/06/2009 2009.

[13] S. Chitta, M. Piccoli, and J. Sturm, “Tactile object class and internal state recognition for mobile manipulation,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2010, pp. 2342–2348.

[14] N. Gorges, S. E. Navarro, D. Göger, and H. Wörn, “Haptic object recognition using passive joints and haptic key features,” in *In Proceedings of the IEEE International Conference on Robotics and Automation*, 2010.

[15] C. Cortes and V. Vapnik, “Support vector networks,” *Machine Learning*, vol. 20, pp. 273–297, 1995.

[16] J. Laaksonen, V. Kyrki, and D. Kragic, “Evaluation of feature representation and machine learning methods in grasp stability learning,” in *10th IEEE-RAS International Conference on Humanoid Robots*, 2010, pp. 112–117.

[17] I. Gaiser, S. Schulz, A. Kargov, H. Klosek, A. Bierbaum, C. Pylatiuk, R. Oberle, T. Werner, T. Asfour, G. Bretthauer, and R. Dillmann, “A new anthropomorphic robotic hand,” in *IEEE-RAS International Conference on Humanoid Robots*, Daejeon, Korea, 2008.

[18] A. Bierbaum, J. Schill, T. Asfour, and R. Dillmann, “Force Position Control for a Pneumatic Anthropomorphic Hand,” in *IEEE-RAS International Conference on Humanoid Robots*, Paris, France, 2009, pp. 21 – 27.

[19] Weiss Robotics, “Tactile sensor module, type: DSA 9335,” accessed 10/09/2009. [Online]. Available: http://www.weiss-robotics.de/

[20] D. Göger, N. Gorges, and H. Wörn, “Tactile Sensing for an Anthropomorphic Robotic Hand: Hardware and Signal Processing,” in *Proc. of the IEEE Int. Conf. on Robotics and Automation, May 12 - 17, 2009, Kobe, Japan*, 2009.

[21] D. Göger and H. Wörn, “A highly versatile and robust tactile sensing system,” in *Proc. of the IEEE Conf. on Sensors*, Atlanta (GA), USA, 2007.

[22] K. Weiss and H. Wörn, “The working principle of resistive tactile sensor cells,” in *Proc. of the IEEE Int. Conf. on Mechatronics and Automation, Canada*, 2005.

Learning shapes as directed closed surfaces
Technical Report

Sandor Szedmak
IIS, University of Innsbruck
sandor.szedmak@uibk.ac.at

Hanchen Xiong
IIS, University of Innsbruck
hanchen.xiong@uibk.ac.at

Justus Piater
IIS, University of Innsbruck
justus.piater@uibk.ac.at

December 19, 2011

1 Introduction

One of the central problems in capturing the environment by a robot is to interpret the objects observed. This interpretation can serve as a starting point to the potential activities. For example: can those objects be grasped, moved and reordered? The interpretation should not stick a label to those objects saying: this is a chair or that is a mug, but it should provide knowledge enough to act in a proper way. A mug or a glass, even a bottle, can be grasped in the same way, thus some common properties are really relevant among those objects, but others, e.g. colors, texture, some details of the shape can be ignored. Some parts, segments of the entire shape of the objects, carry activity-related properties that need to be captured.

The shapes of an object as an entity can not be directly observed by the known machine vision systems. Those systems can yield several different local feature items and the task is to build an abstract shape out of those features. Within that procedure we need to discover how those local items can relate to each other, what is the three dimensional graph connecting them, and recognize those items which can characterize the shape and separate them from those which can relate to something else, e.g. texture.

In learning shapes we assume features which can be characterized by two properties, a 3D position and an orientation, e.g. a surface segment, a patch, etc. These properties can be translated relatively easily into the properties of the potential activities. To collect this kind of features we need proper vision systems which can provide them with sufficient accuracy. Here we assume that these features are available for shape learning.

1.0.1 Shape model

We assume that the shape can be described as a manifold in the three dimensional space. This manifold, we might say surface, is an almost everywhere smooth one allowing to model edges and corners with high curvature, but otherwise it can be partitioned into relatively large connected smooth segments. This assumption expresses the need to eliminate irrelevant small details. Another requirement we should satisfy relates to the potential complexity of the shape, namely it can be a topologically higher order manifold with holes, with a mixture of segments with positive and negative curvature, and convex and concave parts.

To model a surface with complex structure and in the same time forcing a certain high level of smoothness we apply an infinite dimensional parametric representation exploiting the fact that a complex low dimensional manifold can be approximated by a hyperplane in a sufficiently high dimensional space.

The representation space we have chosen is an infinite dimensional Hilbert space of the square integrable functions. Within this space we can apply the probability density functions as features defined on the low dimensional 3D space to be modeled. This mathematical framework allows us to synthesize the probabilistic generative models and the robustness of the maximum margin based discriminative methods. Furthermore, the advantage of the kernel methods in expressing nonlinear relations can be exploited as well. The discrimination happens between the shape and the non-shape points, and the generative, density function based features provide certain local confidence measures on the shape approximation.

The shape modeling is considered as a machine learning procedure where the shape is extracted from local vision features. The learning task is to force a certain type of manifold to closely fit to the parameters, position and orientation, of the visual feature items, and in the same time it has to be as smooth, say simple, as possible which can be achieved via regularization constraining the complexity of the manifold applied.

The outcome of the learning method is an infinite dimensional vector, a combination of probability density functions. This kind of representation admits direct comparison of different objects to express their similarities and dissimilarities. This representation can be reused in other learning method to discover common parts within a given group of object, e.g. by applying Kernel Principal Component Analysis.

The derived shape models can be transformed by any affine transformation, e.g. translation or rotation, via acting on the parameters, expected values and/or covariance matrices, of the density functions used in the expression of the vectors describing the models.

The robot activities, e.g. grasping, can be modeled in a similar framework, thus both the shape models and the actions applied on those shapes as abstract vectors can be located in a common vector space. In this way potential connections between shapes and actions can be predicted for a new shape and for a new action. To do that the relationships between the novel items and the known ones needs to computed in the common vector space.

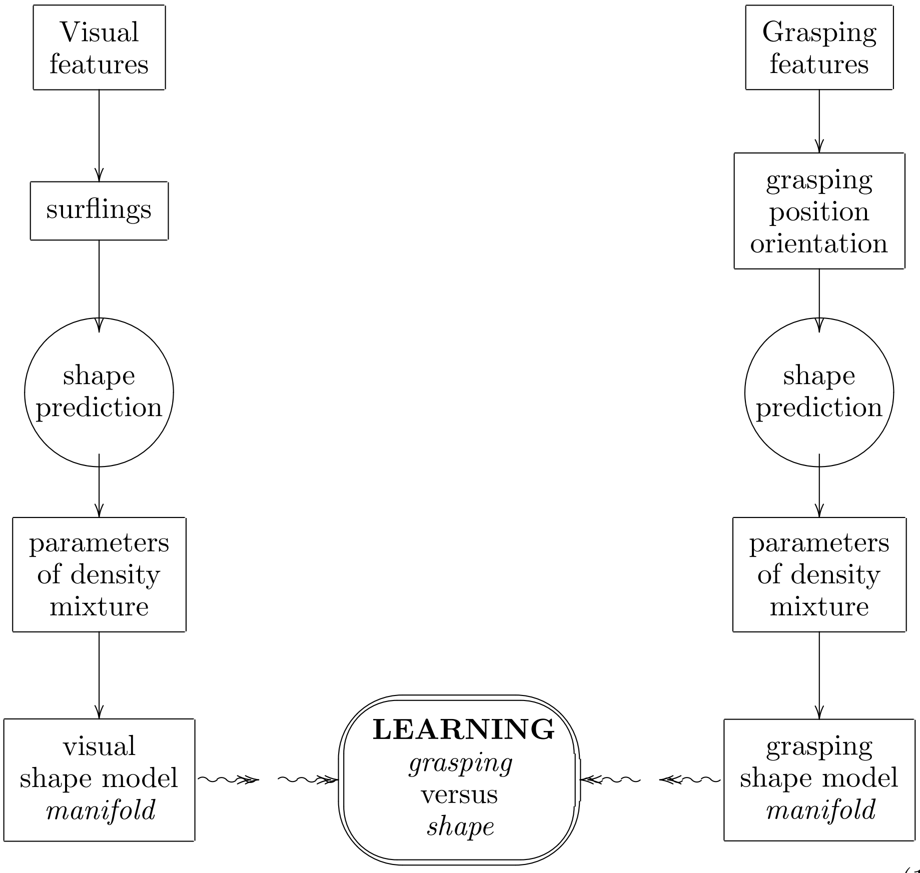
2 Learning task

We are facing the following learning task; given a set of 3D objects characterized by feature representation of different sources

- some visual features, e.g. collection of edges, texlets, surflings, see details about these features in [4],
- grasping properties, e.g. grasp densities,

and based on these data sources we need to learn that how to predict the grasp densities from the visual features. The inverse prediction could provide information to the generalization of the properties of an object to be important in grasping, but in the first case it is not as central as predicting the grasp densities.

To solve this learning task a two-phase model will be introduced; in the first phase so called shape model is computed of a sample of visual features. The parameters derived from the shape model is then used as feature representation of the object to predict the grasp densities. In the first case the shape model can be interpreted as a mixture of density functions, however the optimization framework allows us to further generalize the model, see in Section 5. The shape model can be applied on the grasp densities, since they are given by entities similar to the surflings type visual features, i.e. they are given by position and orientation. The two-phase model is summarized by (1)


Visual features
surflings
shape prediction
parameters of density mixture
visual shape model *manifold*
Grasping features
grasping position orientation
shape prediction
parameters of density mixture
grasping shape model *manifold*
LEARNING *grasping* versus *shape*

(1)

Here we will focus on the surfling type visual features. These features constitute a collection of approximate tangent plane segments of the surface of the 3D objects. The centers and the normal vectors of these segments can be exploited to reproduce the entire surface. In this way the first task in the full learning procedure is to learn these surfaces. To this end we need to create a model of these surfaces such that

- the parameter vectors to be derived of the surface of each object have to live in the same space and in this way they can be compared, and distances, similarity measures can be computed between them,
- the parameter space needs to be sufficiently reach to express the potential complexity of the surfaces,
- since the surflings are only approximation of the real surface, therefore the parameter space should allow to estimate the confidence of the surface model.

To fulfill these requirements the representation space of the surfaces is chosen as a linear vector space containing the probability densities functions.

In this space a surface is expressed as a mixture of densities. The base family of these densities can be chosen as multivariate Gaussians but any other family which allows computationally feasible representation can be considered.

Remark 1. *In this model we are working with linear combination of probability densities that might produce negative probabilities, but this issue is rather technical and will not influence the learning model itself.*

After fitting the surflings based surface model to the objects we can apply the derived surface representations to learn how these features relate to the grasp densities. If in both cases, the surfaces and the grasp densities, are expressed as probability mixture models then the relationships can be revealed not only between the entire models but their parts as well. This kind of analysis can compare the contribution of the elements of the bases spanning the corresponding feature spaces since the elements of these bases, e.g. Gaussian densities, are common among the spaces.

In what follows we assume that there is a preprocessing step of surflings which can separate the surflings of an object from the surflings of the occasional background, hence an object related collection of surflings expresses the properties of an object and only that.

3 Shape model

The shape model of three dimensional objects is built upon the following assumptions:

- The shape $\mathscr{S}$ of an object $\mathscr{O}$ can be expressed by a smooth manifold $\mathcal{M}$ embedded into a Euclidean ambient space $\mathcal{X}$. The dimension of the ambient space is denoted by n.
- The manifold $\mathcal{M}$ is supposed to be closed and all points of the object $\mathscr{S}$ fall inside, with respect to a given orientation of the manifold, or on the manifold.

A smooth manifold can be characterized by an *atlas*, a collection of the pairs $\{U_\alpha, \varphi_\alpha\}$, where $\{U_\alpha\}$, called charts, is a set of open sets covering $\mathcal{M}$, and $\{\varphi_\alpha\}$ is a set of mappings such that for each α $\varphi_\alpha : U_\alpha \to \mathbb{R}^n$, i.e. they map the open sets of the manifold into open sets of a Euclidean space, and these maps and their inverses are differentiable. A surfling at a given point $\mathbf{x}$ of the manifold $\mathcal{M}$ can be interpreted as a segment of the tangent plane at $\mathbf{x}$. A tangent plane is a local linearization of the manifold and it is built upon the charts covering the point $\mathbf{x}$. The definition and properties of the tangent plane can be found for example in [3].

The normal vector to the tangent space can be defined in the ambient space $\mathcal{X}$ as the normal vector of the corresponding tangent plane in $\mathcal{X}$. Another, a more general, way of that which eliminates the need of the ambient space, by considering the normal vector as an element of the linear functionals defined on the tangent plane at a given point of the manifold. The orthogonality then can be expressed by setting the value of these functionals to 0 on all vectors of the corresponding tangent vectors. The elements of the set of these linear functionals are generally called as cotangent vectors.

4 Shape learning

There are given a sample set $\mathcal{S}$ of vector pairs $\{(\mathbf{x}_i, \mathbf{v}_i)\}, i = 1, \dots, m$, where for each i $\mathbf{x}_i \in \mathcal{X} \subset \mathbb{R}^n$, is a vector assigned to a point of the manifold $\mathcal{M}$ in the ambient space $\mathcal{X}$, and $\mathbf{v}_i \in \mathbb{R}^n$ is a vector, the normal vector of the tangent space of $\mathcal{M}$ at $\mathbf{x}_i$ in the ambient space $\mathcal{X}$. We may refer to the pairs $(\mathbf{x}_i, \mathbf{v}_i)$ as surface elements as well. These surface elements are fundamentally based on a sub-sample of charts covering the manifold $\mathcal{M}$

Let $\phi_s : \mathcal{X} \to \mathcal{H}_s$ be a feature representation of the elements of the manifold $\mathcal{M}$ in a Hilbert space $\mathcal{H}_s$. The inner product of $\mathcal{H}_s$ will be expressed by the kernel function $\kappa : \mathcal{H}_s \times \mathcal{H}_s \to \mathbb{R}$. Here we allow to map all vectors of the ambient space of the manifold into the feature space to avoid some technical difficulties.

In the sequel we use the notations $\langle \, , \rangle_X$ and $\langle \, , \rangle_H$ to denote, and to distinct, the inner products in spaces $\mathcal{X}$ and in $\mathcal{H}_s$ respectively. In some cases the subscript is omitted that refers to the inner product in the feature space $\mathcal{H}_s$.

Suppose the manifold $\mathcal{M}$ can be well approximated by surface of $\mathbb{R}^n$, and this surface can be embedded as a hyperplane into the space $\mathcal{H}_s$, thus the implicit function

$$F(x) = \langle \mathbf{u}, \boldsymbol{\phi_s}(x) \rangle_H - 1 = 0 \tag{2}$$

describes the surface, where the vector $\mathbf{u} \in \mathcal{H}_s$ gives the parametrization, and since it is as element of Hilbert space it can be used as identifier of a shape entity.

If the parameter vector $\mathbf{u}$ is given then the manifold can be recovered by inverting the hyperplane from the feature space back into the space $\mathcal{X}$.

4.1 Including normal vectors

To exploit the information coded into normal vectors $\{\mathbf{v}_i\}$ of the surface elements we need to force that for every i the vector $\mathbf{v}_i$ is orthogonal or at least approximately orthogonal to the tangent plane of $\mathcal{M}$ at $\mathbf{x}_i$. If the surface corresponding to the smooth manifold $\mathcal{M}$ is given in $\mathcal{X}$ by the implicit function $F(x) = 0$ then the direction of the normal vector at any $\mathbf{x} \in \mathcal{X}$ is given by the gradient of F

$$\nabla_x F = \left(\frac{\partial F}{\partial x_1}, \ldots, \frac{\partial F}{\partial x_n} \right), \tag{3}$$

where $(x_1, \ldots, x_n)$ the scalar components of the vector $\mathbf{x}$. To force the orthogonality between $\nabla_x F(x)|_{x=x_i}$ and $\mathbf{v}_i$ we have several alternatives, here three of them are enumerated those which lead to linear constraints.

- The following constraints imposes exact orthogonality on the function F
$$\nabla_x F|_{\mathbf{x}=\mathbf{x}_i} = \beta_i \mathbf{v}_i,\ i \in \{1, \ldots, m\} \tag{4}$$
saying the vectors on left and the right hand sides have to be parallel.

- One can eliminate the the coefficients $\{\beta_i\}$ by another form
$$\nabla_x F|_{\mathbf{x}=\mathbf{x_i}} \wedge \mathbf{v}_i = 0\ i \in \{1, \ldots, m\}, \tag{5}$$
where $\wedge$ marks the exterior, sometimes called as wedge, product of two vectors. Here we exploited the fact that the exterior product of any two parallel vectors is equal to $\mathbf{0}$.

- The strict orthogonality constraints can be relaxed by suitable approximations
$$\langle \nabla_x F|_{\mathbf{x}=\mathbf{x}_i}, \mathbf{v}_i \rangle_X \geq D,\ i \in \{1, \ldots, m\}, \tag{6}$$
where D is a positive lower bound of the inner products. Since the inner product can attain its maximum value when the estimated normal vectors of the manifold are parallel to the given set of observed normal vectors, therefore maximizing the lower bound forces the corresponding normal vectors to be closely and uniformly parallel. This type of constraint fundamentally forces the directional derivatives of F in the directions given by $\{\mathbf{v}_i\}$ to be large, and the maximum can be obtained when the vectors are parallel within the inner product.

4.2 Optimization problem

The assumption that the manifold covering an object is closed means that the points of the object is within or on the surface of the volume for which the manifold is the collection of the boundary points. This fact can be expressed as one-class classification problem where the points of the object constituting the class are separated from all other parts of the space containing the object, therefore the surface can be modeled by the following optimization problem:

$$\begin{array}{lll} \min & \frac{1}{2}\|\mathbf{u}\|_2^2 + C_\xi \mathbf{1}'\boldsymbol{\xi} + C_\eta \mathbf{1}'\boldsymbol{\eta} & \\ \text{w.r.t.} & \mathbf{u} \in \mathcal{H}_s,\ \boldsymbol{\xi} \in \mathbb{R}^m,,\ \boldsymbol{\eta} \in \mathbb{R}^m, & \\ \text{s.t.} & \langle \mathbf{u}, \boldsymbol{\phi_s}(\mathbf{x}_i)\rangle_H \geq 1 - \xi_i, & \#\#\text{ fitting the points} \\ & \langle \nabla_x F(x)|_{\mathbf{x}=\mathbf{x}_i}, \mathbf{v}_i\rangle_X \geq E - \eta_i, & \#\#\text{ fitting the normal vectors} \\ & \xi_i \geq 0,\ \eta_i \geq 0,\ i \in \{1, \ldots, m\}, & \end{array} \tag{7}$$

where $E > 0$ is margin scaling parameter for trading between the fitting of the surface points and the normal vectors. This formulation is short symbolic summary of the manifold approximation. To the one-class classification problem one can find introduction in [7] and applications for complex structured learning problems [1], [10], [5] and [11]. An approach similar to that which is presented here is published in [9] and [8]. In the [9] a one-class classification approach is mentioned as well. The main differences between that and our approach can be summarized in two points:

- The incorporation of the normal vectors into the surface approximation is carried out by a maximum margin based regression technique, see further details in Section . This approach allows us to include other characteristic properties of the surface, e.g. curvature, via kernelization.

- The representation of the surface elements is built upon infinite dimensional functional features. This representation can express a probabilistic model which can provide confidence estimation as well.

To transform (7) into a computable form the gradients $\nabla_x F(x)|_{\mathbf{x}=\mathbf{x}_i}$ need to be unfolded. To this end we need to compute the derivative

$$\begin{array}{ll} \nabla_x F(x) & = D_x(F(x))' = D_x(\langle \mathbf{u}, \boldsymbol{\phi_s}(\mathbf{x})\rangle)' \\ & = D_x(\boldsymbol{\phi_s}(\mathbf{x}))'\mathbf{u}. \end{array} \tag{8}$$

If we assume that the dimension of the feature space is finite then $D_x(\boldsymbol{\phi_s}(\mathbf{x}))$ is equal to the Jacobian matrix of the partial derivatives of the vector valued function $\boldsymbol{\phi_s}(\mathbf{x})$ with respect to the vector $\mathbf{x}$. An approach to handling the infinite case is described in Section 4.3.

The primal problem, (7), can be written as

$$\begin{array}{lll} \min & \frac{1}{2}\|\mathbf{u}\|_2^2 + C_\xi \mathbf{1}'\boldsymbol{\xi} + C_\eta \mathbf{1}'\boldsymbol{\eta} & \\ \text{w.r.t.} & \mathbf{u} \in \mathcal{H}_s,\ \boldsymbol{\xi} \in \mathbb{R}^m,,\ \boldsymbol{\eta} \in \mathbb{R}^m, & \\ \text{s.t.} & \langle \mathbf{u}, \boldsymbol{\phi_s}(\mathbf{x}_i)\rangle \geq 1 - \xi_i, & \#\#\text{ fitting the points} \\ & \langle D_x(\boldsymbol{\phi_s}(\mathbf{x}))'|_{x=x_i}\mathbf{u}, \mathbf{v}_i\rangle_X \geq E - \eta_i, & \#\#\text{ fitting the normal vectors} \\ & \xi_i \geq 0,\ \eta_i \geq 0,\ i \in \{1, \ldots, m\}. & \end{array} \tag{9}$$

The solution to this problem can be derived from the Karush-Kuhn-Tucker(KKT) conditions, see details in [2] and references therein. Let the following Lagrangian coefficients be introduced for all i:

$$\begin{array}{lcl} \alpha_i & : & \langle \mathbf{u}, \boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{x}_i) \rangle \geq 1 - \xi_i, \\ \beta_i & : & \langle D_x(\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{x}))'|_{x=x_i} \mathbf{u}, \mathbf{v}_i \rangle_X \geq E - \eta_i, \\ \gamma_i & : & \xi_i \geq 0, \\ \delta_i & : & \eta_i \geq 0 \end{array} \tag{10}$$

Since the constraints are inequalities all Lagrangians have to be nonnegative. The Lagrangian functional of (9) reads as

$$\begin{array}{ll} L(\mathbf{u}, \boldsymbol{\xi}, \boldsymbol{\eta}, \boldsymbol{\alpha}, \boldsymbol{\beta}) & = \frac{1}{2} \langle \mathbf{u}, \mathbf{u} \rangle + C_\xi \mathbf{1}'\boldsymbol{\xi} + C_\eta \mathbf{1}'\boldsymbol{\eta} \\ & - \sum_{i=1}^m \alpha_i \langle \mathbf{u}, \boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{x}_i) \rangle + \sum_{i=1}^m \alpha_i - \sum_{i=1}^m \alpha_i \xi_i \\ & - \sum_{i=1}^m \beta_i \langle D_x(\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{x}))'|_{x=x_i} \mathbf{u}, \mathbf{v}_i \rangle_X + E \sum_{i=1}^m \beta_i - \sum_{i=1}^m \beta_i \eta_i \\ & - \sum_{i=1}^m \gamma_i \xi_i - \sum_{i=1}^m \delta_i \eta_i \\ \text{s.t.} & \alpha_i \geq 0, \ \beta_i \geq 0, \ \gamma_i \geq 0, \ \delta_i \geq 0 \ i = 1, \dots, m. \end{array} \tag{11}$$

The partial derivatives of the Lagrangian with respect to the primal variables and the corresponding KKT conditions are given by

$$\begin{array}{ll} \dfrac{\partial L(\mathbf{u}, \boldsymbol{\xi}, \boldsymbol{\eta}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\delta})}{\partial \mathbf{u}} & = \mathbf{u} - \sum_{i=1}^m \alpha_i \boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{x}_i) - \sum_{i=1}^m \beta_i D_x(\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{x}))'|_{x=x_i} \mathbf{v}_i = 0, \\ \dfrac{\partial L(\mathbf{u}, \boldsymbol{\xi}, \boldsymbol{\eta}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\delta})}{\partial \boldsymbol{\xi}} & = C_\xi \mathbf{1} - \boldsymbol{\alpha} - \boldsymbol{\gamma} = 0, \\ \dfrac{\partial L(\mathbf{u}, \boldsymbol{\xi}, \boldsymbol{\eta}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\delta})}{\partial \boldsymbol{\eta}} & = C_\eta \mathbf{1} - \boldsymbol{\beta} - \boldsymbol{\delta} = 0. \end{array} \tag{12}$$

Thus we have

$$\begin{array}{ll} \mathbf{u} & = \sum_{i=1}^m \alpha_i \boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{x}_i) + \sum_{i=1}^m \beta_i D_x(\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{x}))'|_{x=x_i} \mathbf{v}_i, \\ \boldsymbol{\alpha} & \leq C_\xi \mathbf{1}, \\ \boldsymbol{\beta} & \leq C_\eta \mathbf{1}, \end{array} \tag{13}$$

where in the last two lines the nonnegativity of the components of $\boldsymbol{\gamma}$ and $\boldsymbol{\delta}$ are exploited. After replacing primal variables in the Lagrangian functional with the expressions containing only the Lagrangians we have the dual problem of (9) where the maximization is turned into minimization.

$$\begin{array}{ll} \min & \dfrac{1}{2} \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{bmatrix}' \overbrace{\begin{bmatrix} \mathbf{K}_{\alpha,\alpha} & \mathbf{K}_{\alpha,\beta} \\ \mathbf{K}_{\beta,\alpha} & \mathbf{K}_{\alpha,\alpha} \end{bmatrix}}^{\text{kernel matrix}} \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{bmatrix} - \begin{bmatrix} \mathbf{1} \\ E\mathbf{1} \end{bmatrix}' \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{bmatrix} \\ \text{w.r.t.} & \boldsymbol{\alpha} \in \mathbb{R}_+, \ \boldsymbol{\beta} \in \mathbb{R}_+, \\ \text{s.t.} & \mathbf{0} \leq \boldsymbol{\alpha} \leq C_\xi \mathbf{1}, \\ & \mathbf{0} \leq \boldsymbol{\beta} \leq C_\eta \mathbf{1}, \end{array} \tag{14}$$

The submatrices of the kernel matrix are obtained by

$$\begin{aligned}
(\mathbf{K}_{\alpha,\alpha})_{ij} &= \langle \boldsymbol{\phi_s}(\mathbf{x}_i), \boldsymbol{\phi_s}(\mathbf{x}_j) \rangle\,,\ i,j \in \{1,\dots,m\}, \\
(\mathbf{K}_{\alpha,\beta})_{ij} &= \langle \boldsymbol{\phi_s}(\mathbf{x}_i), D_x(\boldsymbol{\phi_s}(\mathbf{x}))'|_{x=x_j}\mathbf{v}_j \rangle\,,\ i,j \in \{1,\dots,m\}, \\
(\mathbf{K}_{\beta,\alpha})_{ij} &= [\mathbf{K}_{\alpha,\beta}]_{ji},\ i,j \in \{1,\dots,m\}, \\
(\mathbf{K}_{\beta,\beta})_{ij} &= \langle D_x(\boldsymbol{\phi_s}(\mathbf{x}))'|_{x=x_i}\mathbf{v}_i, D_x(\boldsymbol{\phi_s}(\mathbf{x}))'|_{x=x_j}\mathbf{v}_j \rangle\,,\ i,j \in \{1,\dots,m\}.
\end{aligned} \tag{15}$$

4.3 Evaluation of the kernels

When we are going to represent the vectors of the ambient space $\mathcal{X}$ we need to choose a feature space in which a complex surface of $\mathcal{X}$ can be approximated with high fidelity by a hyperplane. A candidate space could be the so called "functional feature" space where each feature vector is represented by a function. These spaces are generally infinite dimensional, thus very high flexibility can be guaranteed.

To realize an infinite dimensional feature space the following construction is proposed. Let $F : \mathcal{X} \times \mathcal{X} \times \Theta \rightarrow \mathbb{R}$ be a real valued function equipped with these properties:

1. F is a nonnegative function,

2. F is square integrable on its full domain,

3. For a fixed $\mathbf{x} \in \mathcal{X}$ and $\theta \in \Theta$

$$\int_{\mathcal{X}} F(\mathbf{t}, \mathbf{x}, \theta) d\mathbf{t} = 1. \tag{16}$$

One can consider function F as a probability density function defined on $\mathcal{X}$ and parametrized on the sets $\mathcal{X}$ and Θ. The parameters taken of $\mathcal{X}$ can be interpreted as localization, e.g. mean, and the parameter θ as scale, e.g. variance. After taking the second and third variables as parameters in F, we can define the following class of functions

$$\mathcal{F} = \{f | f : \mathcal{X} \rightarrow \mathbb{R},\ f(t) = F(t, \mathbf{x}, \theta), \mathbf{t} \in \mathcal{X}, \mathbf{x} \in \mathcal{X}, \theta \in \Theta\}. \tag{17}$$

We might denote these functions for a parameter pair $\mathbf{x}$ and θ by $f(.|\mathbf{x}, \theta)$.

Now the feature mapping is given by

$$\boldsymbol{\phi_s} : \mathcal{X} \rightarrow \mathcal{F}, \tag{18}$$

and defined via the formula

$$\boldsymbol{\phi_s}(\mathbf{x}) = f(.|\mathbf{x}, \theta), \forall \mathbf{x} \in \mathcal{X}, \tag{19}$$

thus the elements of the original ambient space are used as localization of the corresponding density functions, and the scale parameter, θ, is shared among all these densities.

We need to emphasize that the feature mapping is a function valued function.

The value of the function $\boldsymbol{\phi}$ at $\mathbf{t}$ is denoted by $\boldsymbol{\phi}(\mathbf{t}|\mathbf{x})$, where for sake of simplicity the parameter θ which is fixed for all $\mathbf{t}$ and $\mathbf{x}$ is omitted.

4.4 Representation by Gaussian densities

To compute the elements of the kernel matrix in (15) we need to assign concrete representations to the points in the ambient space. Let the feature representation be chosen from the family of the multivariate Gaussian probability density functions

$$\boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}) = f(t|\mathbf{x},\theta) = \frac{1}{(2\pi)^{n/2}\det(\theta)^{1/2}} e^{-\frac{1}{2}(\mathbf{t}-\mathbf{x})'\theta^{-1}(\mathbf{t}-\mathbf{x})}, \tag{20}$$

where $\mathbf{x}$ serves as mean vector and θ as covariance matrix. To force the parsimony of our model the covariance matrix is supposed to be diagonal, and all diagonal elements are equal to σ^2, therefore we have

$$\begin{aligned} \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}) = f(t|\mathbf{x},\theta) &= \frac{1}{(2\pi)^{n/2}\sigma^n} e^{-\frac{1}{2\sigma^2}\langle \mathbf{t}-\mathbf{x},\mathbf{t}-\mathbf{x}\rangle_X} \\ &= \frac{1}{(2\pi)^{n/2}\sigma^n} e^{-\frac{1}{2\sigma^2}\|\mathbf{t}-\mathbf{x}\|2} \\ &= \frac{1}{(2\pi)^{n/2}\sigma^n} e^{-\frac{\sum_{r=1}^{n}(t_r-x_r)^2}{2\sigma^2}}. \end{aligned} \tag{21}$$

The differential operator for general multivariate Gaussian case reads as

$$\begin{aligned} & D_x\boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}) = D_xF(\mathbf{t},\mathbf{x},\theta) = \frac{\partial F(\mathbf{t},\mathbf{x},\theta)}{\partial \mathbf{x}} \\ & = \frac{\partial\left(\frac{1}{(2\pi)^{n/2}\det(\theta)^{1/2}} e^{-\frac{1}{2}(\mathbf{t}-\mathbf{x})'\theta^{-1}(\mathbf{t}-\mathbf{x})}\right)}{\partial \mathbf{x}} \\ & = \frac{1}{(2\pi)^{n/2}\det(\theta)^{1/2}} e^{-\frac{1}{2}(\mathbf{t}-\mathbf{x})'\theta^{-1}(\mathbf{t}-\mathbf{x})} \otimes \theta^{-1}(\mathbf{t}-\mathbf{x}) \\ & = \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}) \otimes \theta^{-1}(\mathbf{t}-\mathbf{x}), \end{aligned} \tag{22}$$

and in case of the reduced diagonal case we have

$$D_x\boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}) = \frac{1}{\sigma^2}\boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}) \otimes (\mathbf{t}-\mathbf{x}), \tag{23}$$

where we need to be aware on the fact that $\boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x})$ is a function of $\mathbf{t}$ as well.

Before the inner products are computed some notations and reformulations are being introduced. For sake of simplicity the following abbreviation is introduced

$$C_G = \frac{1}{(2\pi)^{n/2}\sigma^n}. \tag{24}$$

We are going to exploit the well known identity connecting the tensor and inner products, namely

$$\langle \otimes_{r=1}^{n} \mathbf{u}_r, \otimes_{r=1}^{n} \mathbf{v}_r \rangle = \prod_{r=1}^{n} \langle \mathbf{u}_r, \mathbf{v}_r \rangle \,, \tag{25}$$

further details can be found in Appendix A.

The x_{ir} denotes the rth component of the vector $\mathbf{x}_i$ and similar notation is used for the vectors $\mathbf{v}_i$ as well.

The next simple assertion can eliminate plenty of technical details of the kernel derivation.

Lemma 2. *Assuming that the feature representation given in (21) then the point wise product of any two feature vectors can be expressed as a product of two functions*

$$\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{t}|\mathbf{x}_i)\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{t}|\mathbf{x}_j) = g(\|\mathbf{x}_i, -\mathbf{x}_j\|_2, \sigma) h(\mathbf{t}, (\mathbf{x}_i + \mathbf{x}_j)/2, \sigma/2^{1/2}), \tag{26}$$

such that the function g depends only on the distance $\|\mathbf{x}_i, -\mathbf{x}_j\|_2$ and scale σ but not on $\mathbf{t}$, and the function h is a multivariate Gaussian density function defined on the domain $(\mathbf{t} \in)\mathcal{X}$ with mean $\frac{\mathbf{x}_i, +\mathbf{x}_j}{2}$ and with a diagonal covariance matrix with the same diagonal elements being equal to $\sigma^2/2$.

Proof. The proof is based on a straightforward reformulation of the point wise product, namely

$$\begin{aligned}
&\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{t}|\mathbf{x}_i)\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{t}|\mathbf{x}_j) = C_G^2 e^{-\frac{\|\mathbf{t}-\mathbf{x}_i\|^2}{2\sigma^2}} e^{-\frac{\|\mathbf{t}-\mathbf{x}_j\|^2}{2\sigma^2}} \\
&= C_G^2 e^{-\frac{2\|\mathbf{t}-\frac{\mathbf{x}_i+\mathbf{x}_j}{2})\|^2 + \frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{2}}{2\sigma^2}} \\
&= C_G^2 e^{-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{4\sigma^2}} e^{-\frac{\|\mathbf{t}-\frac{\mathbf{x}_i+\mathbf{x}_j}{2})\|^2}{\sigma^2}} \\
&= C_G^2 \frac{(2\pi)^{n/2}\sigma^n}{2^{n/2}} e^{-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{4\sigma^2}} \frac{2^{n/2}}{(2\pi)^{n/2}\sigma^n} e^{-\frac{\|\mathbf{t}-(\frac{\mathbf{x}_i+\mathbf{x}_j}{2})\|^2}{2(2^{-1/2}\sigma)^2}} \\
&= \underbrace{\frac{1}{(2\pi)^{n/2}(2^{1/2}\sigma)^n} e^{-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{4\sigma^2}}}_{g(\|\mathbf{x}_i,-\mathbf{x}_j\|_2,\sigma)} \underbrace{\frac{2^{n/2}}{(2\pi)^{n/2}\sigma^n} e^{-\frac{\|\mathbf{t}-(\frac{\mathbf{x}_i+\mathbf{x}_j}{2})\|^2}{2(2^{-1/2}\sigma)^2}}}_{h(\mathbf{t},(\mathbf{x}_i,+\mathbf{x}_j)/2,\sigma/2^{1/2})},
\end{aligned} \tag{27}$$

where the last line shows the decomposition claimed. □

From this statement we can conclude that

Corollary 3. *The inner product between any two feature vectors can be computed by*

$$
\begin{aligned}
&\langle \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_i), \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_j)\rangle = \int_{\mathcal{X}} \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_i)\boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_j) d\mathbf{t} \\
&= \tfrac{1}{(2\pi)^{n/2}(2^{1/2}\sigma)^n} e^{-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{4\sigma^2}} \underbrace{\int_{\mathcal{X}} \frac{2^{n/2}}{(2\pi)^{n/2}\sigma^n} e^{-\frac{\|\mathbf{t}-(\frac{\mathbf{x}_i+\mathbf{x}_j}{2})\|^2}{2(2^{-1/2}\sigma)^2}} d\mathbf{t}}_{=1} \\
&= \tfrac{1}{(2\pi)^{n/2}(2^{1/2}\sigma)^n} e^{-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{2(2^{1/2}\sigma)^2}},
\end{aligned}
\tag{28}
$$

which is a Gaussian kernel function with scale, or width, parameter $2(2^{1/2}\sigma)^2$ *multiplied with the scalar* $\frac{1}{(2\pi)^{n/2}(2^{1/2}\sigma)^n}$

It is worth mentioning that the inner product in Corollary 3 can be interpreted as a multivariate Gaussian density function if one of the parameters, $\mathbf{x}_i$ and $\mathbf{x}_j$, is taken as a variable and the other as mean.

4.4.1 Computation of kernel elements

In the derivation of the dual problem we end up with four types of subkernels, see in (15). Two of them are just transpose of each other, thus we need to deal with three types only.

- Based on Corollary 3 we have in the first case

$$
(\mathbf{K}_{\alpha,\alpha})_{ij} = \langle \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_i), \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_j)\rangle = \tfrac{1}{(2\pi)^{n/2}(2^{1/2}\sigma)^n} e^{-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{2(2^{1/2}\sigma)^2}}. \tag{29}
$$

- The cross kernels between the two types of constraints relating to the positions and the surface normals can be computed by

$$
\begin{aligned}
&(\mathbf{K}_{\alpha,\beta})_{ij} = \langle \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_i), D_x(\boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}))'|_{x=x_j}\mathbf{v}_j\rangle \\
&= \langle \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_i), \frac{1}{\sigma^2}\left[\boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_j) \otimes (\mathbf{t}-\mathbf{x}_j)'\right]\mathbf{v}_j\rangle \\
&= \frac{1}{\sigma^2}\langle \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_i), \langle(\mathbf{t}-\mathbf{x}_j), \mathbf{v}_j\rangle_X \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_j)\rangle \\
&= \frac{1}{\sigma^2}\langle \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_i), (\langle\mathbf{t}, \mathbf{v}_j\rangle_X - \langle\mathbf{x}_j, \mathbf{v}_j\rangle_X)\boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_j)\rangle \\
&= \frac{1}{\sigma^2}\big(\langle \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_i), \langle\mathbf{t}, \mathbf{v}_j\rangle_X \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_j)\rangle - \langle \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_i), \langle\mathbf{x}_j, \mathbf{v}_j\rangle_X)\boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_j)\rangle\big).
\end{aligned}
\tag{30}
$$

In computing this expression by parts we can exploit Corollary 3 again

$$
\begin{aligned}
&\langle \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_i), \langle\mathbf{x}_j, \mathbf{v}_j\rangle_X \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_j)\rangle = \langle\mathbf{x}_j, \mathbf{v}_j\rangle_X \int_{\mathcal{X}} \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_i)\boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_j) d\mathbf{t} \\
&= \langle\mathbf{x}_j, \mathbf{v}_j\rangle_X \tfrac{1}{(2\pi)^{n/2}(2^{1/2}\sigma)^n} e^{-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{2(2^{1/2}\sigma)^2}},
\end{aligned}
\tag{31}
$$

and

$$\begin{aligned} &\langle \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_i), \langle \mathbf{t}, \mathbf{v}_j \rangle_X \, \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_j) \rangle = \int_{\mathcal{X}} \langle \mathbf{t}, \mathbf{v}_j \rangle_X \, \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_i) \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_j) d\mathbf{t} \\ &= \langle \mathbf{v}_j, \int_{\mathcal{X}} \mathbf{t} \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_i) \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_j) d\mathbf{t} \rangle_X . \end{aligned} \tag{32}$$

After applying the decomposition of Lemma 2 note that the expression in the integral can be interpreted as an expected value computation

$$\begin{aligned} &\langle \mathbf{v}_j, \int_{\mathcal{X}} \mathbf{t} \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_i) \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_j) d\mathbf{t} \rangle_X \\ &= g(\|\mathbf{x}_i, -\mathbf{x}_j\|_2, \sigma) \, \langle \mathbf{v}_j, \int_{\mathcal{X}} \mathbf{t} h(\mathbf{t}, (\mathbf{x}_i, +\mathbf{x}_j)/2, \sigma/2^{1/2}) d\mathbf{t} \rangle_X \\ &= g(\|\mathbf{x}_i, -\mathbf{x}_j\|_2, \sigma) \, \langle \mathbf{v}_j, (\mathbf{x}_i, +\mathbf{x}_j)/2 \rangle_X \\ &= \frac{1}{(2\pi)^{n/2}(2^{1/2}\sigma)^n} e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2(2^{1/2}\sigma)^2}} \, \langle \mathbf{v}_j, \frac{\mathbf{x}_i + \mathbf{x}_j}{2} \rangle_X . \end{aligned} \tag{33}$$

Then putting together the sub-expressions an element of the cross kernel is given by

$$(\mathbf{K}_{\alpha,\beta})_{ij} = \frac{1}{\sigma^2} \frac{1}{(2\pi)^{n/2}(2^{1/2}\sigma)^n} e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2(2^{1/2}\sigma)^2}} \, \langle \mathbf{v}_j, \frac{\mathbf{x}_i - \mathbf{x}_j}{2} \rangle_X . \tag{34}$$

- We have also the transpose of the previously computed sub-kernel.

$$(\mathbf{K}_{\beta,\alpha})_{ij} = [\mathbf{K}_{\alpha,\beta}]_{ji} = \frac{1}{\sigma^2} \frac{1}{(2\pi)^{n/2}(2^{1/2}\sigma)^n} e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2(2^{1/2}\sigma)^2}} \, \langle \mathbf{v}_i, \frac{\mathbf{x}_j - \mathbf{x}_i}{2} \rangle_X . \tag{35}$$

- The computation of kernel items relating to the normal vectors follows a schema resembling to those mentioned above.

$$\begin{aligned} &(\mathbf{K}_{\beta,\beta})_{ij} = \langle D_x(\boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}))'|_{x=x_i} \mathbf{v}_i, D_x(\boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}))'|_{x=x_j} \mathbf{v}_j \rangle \\ &= \frac{1}{\sigma^4} \langle \langle (\mathbf{t} - \mathbf{x}_i), \mathbf{v}_i \rangle_X \, \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_i), \langle (\mathbf{t} - \mathbf{x}_j), \mathbf{v}_j \rangle_X \, \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_j) \rangle \\ &= \frac{1}{\sigma^4} \langle \langle (\mathbf{t} - \mathbf{x}_i), \mathbf{v}_i \rangle_X \, \langle (\mathbf{t} - \mathbf{x}_j), \mathbf{v}_j \rangle_X \, \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_i), \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_j) \rangle \\ &= \frac{1}{\sigma^4} \Big[\langle \langle \mathbf{t}, \mathbf{v}_i \rangle_X \, \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_i), \langle \mathbf{t}, \mathbf{v}_j \rangle_X \, \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_j) \rangle \\ &\quad + \langle \langle \mathbf{x}_i, \mathbf{v}_i \rangle_X \, \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_i), \langle \mathbf{x}_j, \mathbf{v}_j \rangle_X \, \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_j) \rangle \\ &\quad - \langle \langle \mathbf{t}, \mathbf{v}_i \rangle_X \, \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_i), \langle \mathbf{x}_j, \mathbf{v}_j \rangle_X \, \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_j) \rangle \\ &\quad - \langle \langle \mathbf{x}_i, \mathbf{v}_i \rangle_X \, \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_i), \langle \mathbf{t}, \mathbf{v}_j \rangle_X \, \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_j) \rangle \Big]. \end{aligned} \tag{36}$$

Except the first term we can apply almost the same unfolding steps on the sub-expressions that have been used above thus we have

$$\begin{aligned} &\langle \langle \mathbf{x}_i, \mathbf{v}_i \rangle_X \, \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_i), \langle \mathbf{x}_j, \mathbf{v}_j \rangle_X \, \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_j) \rangle \\ &\quad = \langle \mathbf{x}_i, \mathbf{v}_i \rangle_X \, \langle \mathbf{x}_j, \mathbf{v}_j \rangle_X \, \langle \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_i), \boldsymbol{\phi_s}(\mathbf{t}|\mathbf{x}_j) \rangle \\ &\quad = \langle \mathbf{x}_i, \mathbf{v}_i \rangle_X \, \langle \mathbf{x}_j, \mathbf{v}_j \rangle_X \, \frac{1}{(2\pi)^{n/2}(2^{1/2}\sigma)^n} e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2(2^{1/2}\sigma)^2}} , \end{aligned} \tag{37}$$

$$\begin{aligned}&\langle\langle\mathbf{t},\mathbf{v}_i\rangle_X\,\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{t}|\mathbf{x}_i),\langle\mathbf{x}_j,\mathbf{v}_j\rangle_X\,\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{t}|\mathbf{x}_j)\rangle\\&\quad=\langle\mathbf{x}_j,\mathbf{v}_j\rangle_X\,\langle\langle\mathbf{t},\mathbf{v}_i\rangle_X\,\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{t}|\mathbf{x}_i),\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{t}|\mathbf{x}_j)\rangle\\&\quad=\langle\mathbf{x}_j,\mathbf{v}_j\rangle_X\,\langle\tfrac{\mathbf{x}_i+\mathbf{x}_j}{2},\mathbf{v}_i\rangle_X\,\frac{1}{(2\pi)^{n/2}(2^{1/2}\sigma)^n}e^{-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{2(2^{1/2}\sigma)^2}}\end{aligned}\tag{38}$$

and

$$\begin{aligned}&\langle\langle\mathbf{x}_i,\mathbf{v}_i\rangle_X\,\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{t}|\mathbf{x}_i),\langle\mathbf{t},\mathbf{v}_j\rangle_X\,\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{t}|\mathbf{x}_j)\rangle\\&\quad=\langle\mathbf{x}_i,\mathbf{v}_i\rangle_X\,\langle\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{t}|\mathbf{x}_i),\langle\mathbf{t},\mathbf{v}_j\rangle_X\,\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{t}|\mathbf{x}_j)\rangle\\&\quad=\langle\mathbf{x}_i,\mathbf{v}_i\rangle_X\,\langle\tfrac{\mathbf{x}_i+\mathbf{x}_j}{2},\mathbf{v}_j\rangle_X\,\frac{1}{(2\pi)^{n/2}(2^{1/2}\sigma)^n}e^{-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{2(2^{1/2}\sigma)^2}}.\end{aligned}\tag{39}$$

The first term requires a little bit more care, where we have

$$\begin{aligned}&\langle\langle\mathbf{t},\mathbf{v}_i\rangle_X\,\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{t}|\mathbf{x}_i),\langle\mathbf{t},\mathbf{v}_j\rangle_X\,\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{t}|\mathbf{x}_j)\rangle\\&\quad=\textstyle\int_{\mathcal{X}}\langle\mathbf{t},\mathbf{v}_i\rangle_X\,\langle\mathbf{t},\mathbf{v}_j\rangle_X\,\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{t}|\mathbf{x}_i)\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{t}|\mathbf{x}_j)d\mathbf{t}.\end{aligned}\tag{40}$$

If we apply the decomposition of Lemma 2 again and the identity relating to the inner product of tensor products, see (25), we receive the following chain of equalities

$$\begin{aligned}&\langle\langle\mathbf{t},\mathbf{v}_i\rangle_X\,\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{t}|\mathbf{x}_i),\langle\mathbf{t},\mathbf{v}_j\rangle_X\,\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{t}|\mathbf{x}_j)\rangle\\&=\langle(\mathbf{v}_i\otimes\mathbf{v}_j),\textstyle\int_{\mathcal{X}}(\mathbf{t}\otimes\mathbf{t})\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{t}|\mathbf{x}_i)\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{t}|\mathbf{x}_j)d\mathbf{t}\rangle_{\mathrm{Frob}}\\&=\langle(\mathbf{v}_i\otimes\mathbf{v}_j),\textstyle\int_{\mathcal{X}}(\mathbf{t}\otimes\mathbf{t})g(\|\mathbf{x}_i,-\mathbf{x}_j\|_2,\sigma)h(\mathbf{t},(\mathbf{x}_i,+\mathbf{x}_j)/2,\sigma/2^{1/2})d\mathbf{t}\rangle_{\mathrm{Frob}}\\&=g(\|\mathbf{x}_i,-\mathbf{x}_j\|_2,\sigma)\,\langle(\mathbf{v}_i\otimes\mathbf{v}_j),\textstyle\int_{\mathcal{X}}(\mathbf{t}\otimes\mathbf{t})h(\mathbf{t},(\mathbf{x}_i,+\mathbf{x}_j)/2,\sigma/2^{1/2})d\mathbf{t}\rangle_{\mathrm{Frob}}.\end{aligned}\tag{41}$$

Note the integral expression is equal to the second, non-centralized, moment of the multivariate Gaussian variable with density function h. Based on the identity

$$\mathrm{cov}(\mathbf{t})=E(\mathbf{t}\otimes\mathbf{t})-E(\mathbf{t})\otimes E(\mathbf{t})\tag{42}$$

which displays that how the covariance can be expressed by the first the second moments of vector valued random variables, thus we can write

$$\begin{aligned}&\langle\langle\mathbf{t},\mathbf{v}_i\rangle_X\,\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{t}|\mathbf{x}_i),\langle\mathbf{t},\mathbf{v}_j\rangle_X\,\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{t}|\mathbf{x}_j)\rangle\\&=g(\|\mathbf{x}_i,-\mathbf{x}_j\|_2,\sigma)\,\langle(\mathbf{v}_i\otimes\mathbf{v}_j),\mathrm{cov}(\mathbf{t})+E(\mathbf{t})\otimes E(\mathbf{t})\rangle_{\mathrm{Frob}}\\&=g(\|\mathbf{x}_i,-\mathbf{x}_j\|_2,\sigma)\,\langle(\mathbf{v}_i\otimes\mathbf{v}_j),\tfrac{\sigma^2}{2}\mathbf{I}_n+\tfrac{\mathbf{x}_i,+\mathbf{x}_j}{2}\otimes\tfrac{\mathbf{x}_i,+\mathbf{x}_j}{2}\rangle_{\mathrm{Frob}},\end{aligned}\tag{43}$$

where $\mathbf{I}_n$ denotes the n-dimensional identity matrix. Now we can reverse Identity 25

$$\begin{aligned}&\langle\langle\mathbf{t},\mathbf{v}_i\rangle_X\,\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{t}|\mathbf{x}_i),\langle\mathbf{t},\mathbf{v}_j\rangle_X\,\boldsymbol{\phi}_{\boldsymbol{s}}(\mathbf{t}|\mathbf{x}_j)\rangle\\&=g(\|\mathbf{x}_i,-\mathbf{x}_j\|_2,\sigma)(\tfrac{\sigma^2}{2}\,\langle\mathbf{v}_i,\mathbf{v}_j\rangle_X+\langle\mathbf{v}_i,\tfrac{\mathbf{x}_i,+\mathbf{x}_j}{2}\rangle_X\,\langle\mathbf{v}_j,\tfrac{\mathbf{x}_i,+\mathbf{x}_j}{2}\rangle\\&=(\tfrac{\sigma^2}{2}\,\langle\mathbf{v}_i,\mathbf{v}_j\rangle_X+\langle\mathbf{v}_i,\tfrac{\mathbf{x}_i,+\mathbf{x}_j}{2}\rangle_X\,\langle\mathbf{v}_j,\tfrac{\mathbf{x}_i,+\mathbf{x}_j}{2}\rangle_X)\frac{1}{(2\pi)^{n/2}(2^{1/2}\sigma)^n}e^{-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{2(2^{1/2}\sigma)^2}}.\end{aligned}\tag{44}$$

After combining the sub-expressions we arrive at

$$
\begin{aligned}
(\mathbf{K}_{\beta,\beta})_{ij} &= \tfrac{1}{\sigma^4}\tfrac{1}{(2\pi)^{n/2}(2^{1/2}\sigma)^n}e^{-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{4\sigma^2}} \\
&\quad \Big(\langle \mathbf{x}_i, \mathbf{v}_i\rangle_X \langle \mathbf{x}_j, \mathbf{v}_j\rangle_X - \langle \mathbf{x}_j, \mathbf{v}_j\rangle_X \langle \tfrac{\mathbf{x}_i+\mathbf{x}_j}{2}, \mathbf{v}_i\rangle_X - \langle \mathbf{x}_i, \mathbf{v}_i\rangle_X \langle \tfrac{\mathbf{x}_i+\mathbf{x}_j}{2}, \mathbf{v}_j\rangle_X \\
&\qquad + \tfrac{\sigma^2}{2}\langle \mathbf{v}_i, \mathbf{v}_j\rangle_X + \langle \mathbf{v}_i, \tfrac{\mathbf{x}_i,+\mathbf{x}_j}{2}\rangle_X \langle \mathbf{v}_j, \tfrac{\mathbf{x}_i,+\mathbf{x}_j}{2}\rangle_X \Big) \\
&= \tfrac{1}{\sigma^4}\tfrac{1}{(2\pi)^{n/2}(2^{1/2}\sigma)^n}e^{-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{2(2^{1/2}\sigma)^2}} (\tfrac{\sigma^2}{2}\langle \mathbf{v}_i, \mathbf{v}_j\rangle_X + \langle \tfrac{\mathbf{x}_j-\mathbf{x}_i}{2}, \mathbf{v}_j\rangle_X \langle \tfrac{\mathbf{x}_i-\mathbf{x}_j}{2}, \mathbf{v}_i\rangle_X).
\end{aligned} \tag{45}
$$

5 General description of the learning model

The learning task that we are going to solve is the following. There is a set, called sample, of pairs of output and input objects $\{(y_i, x_i) : y_i \in \mathcal{Y},\ x_i \in \mathcal{X},\ i = 1, \dots, m,\}$ independently and identically chosen out of an unknown multivariate distribution $\mathcal{P}(Y, X)$. Here we would like to emphasize the input and output objects can be arbitrary, e.g. they may be graphs, matrices, functions, probability distributions etc. To these objects there are given two functions $\phi : \mathcal{X} \to \mathcal{H}_\phi$ and $\psi : \mathcal{Y} \to \mathcal{H}_\psi$ mapping the input and output objects respectively into linear vector spaces, called in the sequel, feature space in case of the inputs and label space when the outputs are considered.

The objective is to find a linear function acting on the feature space

$$
f(\boldsymbol{\phi}(x)) = \mathbf{W}\boldsymbol{\phi}(x) + \mathbf{b} \tag{46}
$$

and produces a prediction of every input object in the label space and in this way could implicitly give back a corresponding output object. Formally we have

$$
y = \psi^{-1}(\psi(y)) = \psi^{-1}(f(\boldsymbol{\phi}(x))). \tag{47}
$$

The learning procedure can be summarized by the following table:

Embedding	$\phi : \overbrace{\text{input space}}^{\mathcal{X}} \rightarrow \overbrace{\text{feature space}}^{\mathcal{H}_\phi},$ $\psi : \overbrace{\text{output space}}^{\mathcal{Y}} \rightarrow \overbrace{\text{label space}}^{\mathcal{H}_\psi},$
Similarity transformation	$\widetilde{\mathbf{W}} = (\mathbf{W}, \mathbf{b}) \Rightarrow \boldsymbol{\psi}(y) \sim \widetilde{\mathbf{W}}\boldsymbol{\phi}(x),$
Inversion	$\boldsymbol{\psi}^{-1} : \overbrace{\text{label space}}^{\mathcal{H}_\psi} \rightarrow \overbrace{\text{output space}}^{\mathcal{Y}}.$

In the framework of the Support Vector Machine the outputs represent two classes and the labels are chosen out of the set $y_i \in \{-1, +1\}$. The aim is to find a separating hyperplane, via its normal vector, such that the distance between the elements of the two classes, called margin, is the possible largest measured in the direction of this normal vector. This base schema can be extended allowing some sample items to fall closer to the separating hyperplane than the margin. This is demonstrated on Figure 1

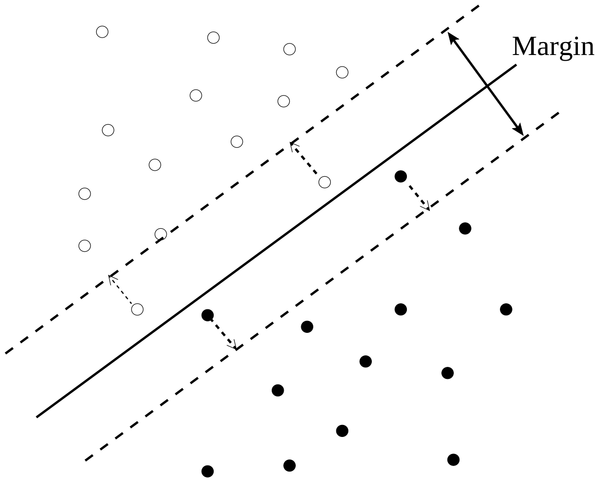


Figure 1: The schema of the Support Vector Machine. There are two classes that we are going to separate by using a hyperplane maximizing the distance between the classes and minimizing the potential errors

This learning scenario can be formulated as an optimization problem similar to this:

$$\begin{array}{ll} \min & \frac{1}{2}\boxed{\mathbf{w}'\mathbf{w}} + C\mathbf{1}'\boldsymbol{\xi} \\ & \\ \text{w.r.t.} & \boxed{\mathbf{w} : \mathcal{H}_\phi \to \mathbb{R}},\ \text{normal vector} \\ & \boxed{b \in \mathbb{R}},\ \text{bias} \\ & \boldsymbol{\xi} \in \mathbb{R}^m,\ \text{error vector} \\ & \\ \text{s.t.} & \boxed{y_i(\mathbf{w}'\boldsymbol{\phi}(\mathbf{x}_i) + b)} \geq 1 - \xi_i \\ & \boldsymbol{\xi} \geq \mathbf{0},\ i = 1, \dots, m. \end{array}$$

5.1 Reinterpretation of the normal vector w

The normal vector **w** formally behaves as a linear transformation acting on the feature vectors which makes rise the idea to extend the capability of the original schema. This reinterpretation can be characterized briefly in the following way

SVM	**ExtendedView**
• $\mathbf{w}$ is the normal vector of the separating hyperplane.	• $\mathbf{W}$ is a linear operator projecting the feature space into the label space.
• $y_i \in \{-1, +1\}$ binary outputs.	• $y_i \in \mathcal{Y}$ arbitrary outputs
• The labels are equal to the binary objects.	• $\boldsymbol{\psi}(y_i) \in \mathcal{H}_\psi$ are the labels, the embedded outputs in a linear vector space

If we apply a one-dimensional normalized label space invoking binary labels $\{-1, +1\}$ in the general framework one can restore the original scenario of the SVM, and the normal vector is a projection into the one dimensional label space.

The extended form of the SVM tries to find an affine transformation which maps the configuration of the input items to gain the highest similarity between the image of the inputs and the outputs.

In summarizing the learning task we end up in the following optimization problem presented parallel with the original primal form of the SVM to emphasize the similarities and dissimilarities between the original and the extended form.

Primal problems for maximum margin learning

	Binary class learning	Vector label learning
	Support Vector Machine(SVM)	Maximum Margin Regression(MMR)
min	$\frac{1}{2}\underbrace{\boxed{\mathbf{w}'\mathbf{w}}}_{\\|\mathbf{w}\\|_2^2} + C\mathbf{1}'\boldsymbol{\xi}$	$\frac{1}{2}\underbrace{\boxed{\mathbf{tr}(\mathbf{W}'\mathbf{W})}}_{\\|\mathbf{W}\\|_F^2} + C\mathbf{1}'\boldsymbol{\xi}$
w.r.t.	$\boxed{\mathbf{w} : \mathcal{H}_\phi \rightarrow \mathbb{R},}$ normal vec. $\boxed{b \in \mathbb{R},}$ bias, $\boldsymbol{\xi} \in \mathbb{R}^m,$ error vector,	$\boxed{\mathbf{W} : \mathcal{H}_\phi \rightarrow \mathcal{H}_\psi,}$ linear operator, $\boxed{\mathbf{b} \in \mathcal{H}_\psi,}$ translation(bias), $\boldsymbol{\xi} \in \mathbb{R}^m,$ error vector,
s.t.	$\boxed{y_i(\mathbf{w}'\boldsymbol{\phi}(\mathbf{x}_i) + b)} \geq 1 - \xi_i,$ $\boldsymbol{\xi} \geq \mathbf{0},\ i = 1, \dots, m,$	$\boxed{\langle \boldsymbol{\psi}(\mathbf{y}_i), \mathbf{W}\boldsymbol{\phi}(\mathbf{x}_i) + \mathbf{b}\rangle_{\mathcal{H}_\psi}} \geq 1 - \xi_i,$ $\boldsymbol{\xi} \geq \mathbf{0},\ i = 1, \dots, m.$

In the extended formulation we exploit the fact the Frobenius norm and the Frobenius inner product correspond to the linear vector space of matrices with the dimension being equal to the number of elements of the matrices, hence it gives an isomorphism between the space spanned by the normal vector of the hyperplane occurring in the SVM and the space spanned by the linear transformations.

One can recognize that if no bias term included in the MMR problem then we have a completely symmetric relationship between the label and the feature space via the representations of the input and the output items, namely

$$\langle \boldsymbol{\psi}(\mathbf{y}_i), \mathbf{W}\boldsymbol{\phi}(\mathbf{x}_i)\rangle_{\mathcal{H}_\psi} = \langle \mathbf{W}^*\boldsymbol{\psi}(\mathbf{y}_i), \boldsymbol{\phi}(\mathbf{x}_i)\rangle_{\mathcal{H}_\phi} = \langle \boldsymbol{\phi}(\mathbf{x}_i), \mathbf{W}^*\boldsymbol{\psi}(\mathbf{y}_i)\rangle_{\mathcal{H}_\phi}\,.$$

Thus, in predicting the input items as the image of the corresponding linear function defined on the outputs the adjoint of $\mathbf{W}$, $\mathbf{W}^*$, need to be used. This adjoint is equal to the transpose of the matrix representation of $\mathbf{W}$ when both the label space and the feature space are finite.

5.2 Dual problem

The dual problem of the MMR presented in the right column of (5.1) is given by

$$\begin{array}{ll}
\min & \sum_{i,j=1}^m \alpha_i\alpha_j \overbrace{\langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j)\rangle}^{\kappa_{ij}^{\phi}} \overbrace{\langle \boldsymbol{\psi}(\mathbf{y}_i), \boldsymbol{\psi}(\mathbf{y}_j))\rangle}^{\kappa_{ij}^{\psi}} - \sum_{i=1}^m \alpha_i, \\
\text{w.r.t.} & \alpha_i \in \mathbb{R}, \\
\text{s.t.} & \sum_{i=1}^m (\boldsymbol{\psi}(\mathbf{y}_i))_t\alpha_i = 0,\ t = 1,\dots,\dim(\mathcal{H}_\psi), \\
 & 0 \le \alpha_i \le C,\ i = 1,\dots,m. \\
\hline
\kappa_{ij}^{\phi} & \text{kernel items corresponding to the feature vectors,} \\
\kappa_{ij}^{\psi} & \text{kernel items corresponding to the label vectors}
\end{array}$$

The objective function contains no direct reference to the implicit representation either the label or the feature vectors, only the corresponding kernel elements appear. The symmetry of the objective function is clearly recognizable showing that the underlying problem without bias is completely reversible.

The constraints

$$\sum_{i=1}^m (\boldsymbol{\psi}(\mathbf{y}_i))_t\alpha_i = 0,\ t = 1,\dots,\dim(\mathcal{H}_\psi) \tag{48}$$

appear in the dual only if the bias term is included into the primal model.

The explicit occurrences of the label vectors can be transformed into implicit ones by exploiting that the feasibility domain covered by the constraints:

$$\sum_{i=1}^m (\boldsymbol{\psi}(\mathbf{y}_i))_t\alpha_i = 0,\ t = 1,\dots,\dim(\mathcal{H}_\psi),$$

coincides with a domain

$$\sum_{i=1}^{m} \kappa_{ij}^{\psi} \alpha_i = 0, \ j = 1, \ldots, m$$

referring only to inner products of the label vectors.

5.3 Simple solution for the unbiased case

The unbiased case of has the form

$$\begin{array}{ll} \min & \frac{1}{2}\boldsymbol{\alpha}'\mathbf{K}\boldsymbol{\alpha}' + \mathbf{q}'\boldsymbol{\alpha} \\ \text{w.r.t.} & \boldsymbol{\alpha}, \\ \text{s.t.} & \mathbf{0} \leq \boldsymbol{\alpha} \leq C, \end{array} \tag{49}$$

where $\mathbf{K} = \mathbf{K}_{\psi(y)} \bullet \mathbf{K}_{\phi(x)}$ is the point wise product of the output and input kernel, and $\mathbf{q} = \mathbf{1}$ a vector with every component equals to 1

The next simple, coordinate descent, approach seems to be over simplified but when the sample size is really large, > 10000 then the inherent simplicity becomes superior when the matrix Q is dense. We would like to emphasize another approach, e.g. interior point methods, could perform better in smaller problems, but the difference not much significant.

Step 1 Let $\boldsymbol{\alpha}^{\mathbf{0}} = \mathbf{0}$ a feasible initial solution, ϵ_α an error tolerance, and $k = 0$ a counter.

Step 2 $k = k + 1$, $\boldsymbol{\alpha}^k = \boldsymbol{\alpha}^{k-1}$, and set the component index of $\boldsymbol{\alpha}^k$, i to 0.

Step 3 Solve the unconditional problem:

$$\min_\tau (\boldsymbol{\alpha}^k + \mathbf{e}_i\tau)'\mathbf{K}(\boldsymbol{\alpha}^k + \mathbf{e}_i\tau) + \mathbf{q}'(\boldsymbol{\alpha}^k + \mathbf{e}_i\tau), \tag{50}$$

where $\mathbf{e}_i$ is a vector with 0 components except the component i which is equal to 1. Problem (50) has a closed form optimal solution τ_* which reads as

$$\tau_* = \frac{-q_i - \mathbf{e}_i'\mathbf{K}\boldsymbol{\alpha}^k}{Q_{ii}} = \frac{-q_i - \mathbf{K}_i\boldsymbol{\alpha}^k}{Q_{ii}}, \tag{51}$$

where $\mathbf{K}_i$ denotes the ith row of Q.

Step 4 Set the ith component of $\boldsymbol{\alpha}^k$ to $\alpha_i^k = \alpha_i^k + \tau$.

Step 5 If $\alpha_i^k > C$ then $\alpha_i^k = C$, and if $\alpha_i^k < 0$ then $\alpha_i^k = 0$; which operations is the projection of an infeasible solution back into(onto) the domain of the box constraint.

Step 6 $i = i + 1$, go to Step 3!

Step 7 If $\|\boldsymbol{\alpha}^k - \boldsymbol{\alpha}^{k-1}\|_2^2 \leq \epsilon_\alpha$ then Stop, otherwise go to Step 2!

The reasonable advantage of this coordinate descent method is that: it requires only a row of the matrix $\mathbf{K}$ in an iteration step, furthermore the division by Q_{ii} is a numerically well controllable operation, since Q_{ii} has a constant value during the procedure and if the kernels are normalized it has value 1 eliminating the need of any division in the computation process.

5.4 Prediction

After solving the dual problem with the help of the optimum dual variables we can write up the optimal linear operator

$$\mathbf{W} \quad = \textstyle\sum_{i=1}^{m} \alpha_i \boldsymbol{\psi}(\mathbf{y}_i)\boldsymbol{\phi}(\mathbf{x}_i)'.$$

Comparing this expression with the corresponding formula which gives the optimal solution to the SVM, i.e.

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \boldsymbol{\phi}(\mathbf{x}_i),$$

we can see that the new part includes the vectors representing the output items which in the SVM were only scalar values but we could say in the new interpretation they are one-dimensional vectors. With the expression of the linear operator $\mathbf{W}$ at hand the prediction to a new input item $\mathbf{x}$ can be written up by

$$\begin{aligned} \boldsymbol{\psi}(\mathbf{y}) \quad &= \mathbf{W}\boldsymbol{\phi}(\mathbf{x}) \\ &= \textstyle\sum_{i=1}^{m} \alpha_i \boldsymbol{\psi}(\mathbf{y}_i) \boxed{\underbrace{\langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}) \rangle}_{\kappa^{\phi}(\mathbf{x}_i,\mathbf{x})}}. \end{aligned}$$

It involves only the input kernel κ^{ϕ} and provides the implicit representation of the prediction $\boldsymbol{\psi}(\mathbf{y})$ to the corresponding output $\mathbf{y}$.

If only the implicit image of the output is given we need to invert the function ψ to gain the $\mathbf{y}$. This inversion problem is sometimes called as pre-image problem as well. Unfortunately there is no general procedure to do that efficiently in case of complex and non-invertible mapping. We mention here a schema that can be applied when the set of all possible outputs is finite with a reasonable small cardinality. The meaning of the “reasonable small” cardinality depends on the given problem, e.g. how expensive to compute the inner product between the output items in the label space where they are represented.

At the conditions mentioned above we can follow this scenario

$$\begin{aligned}
\mathbf{y} \quad & \in \widetilde{\mathcal{Y}} \quad \Leftarrow \text{Set of the possible outputs},\\
\mathbf{y}^* \quad & = \arg\max_{\mathbf{y}\in\widetilde{\mathcal{Y}}} \boldsymbol{\psi}(\mathbf{y})'\mathbf{W}\boldsymbol{\phi}(\mathbf{x}),\\
& = \arg\max_{\mathbf{y}\in\widetilde{\mathcal{Y}}} \sum_{i=1}^m \alpha_i \overbrace{\langle \boldsymbol{\psi}(\mathbf{y}), \boldsymbol{\psi}(\mathbf{y}_i)\rangle}^{\kappa^{\psi}(\mathbf{y},\mathbf{y}_i)} \overbrace{\langle \boldsymbol{\phi}(\mathbf{x}_i)'\boldsymbol{\phi}(\mathbf{x})\rangle}^{\kappa^{\phi}(\mathbf{x}_i,\mathbf{x})},\\
\mathbf{y} \quad & \in \widetilde{\mathcal{Y}} = \{\mathbf{y}_1, \dots, \mathbf{y}_K\},\ K \ll \infty.
\end{aligned}$$

The main advantage of this approach is that it requires only the inner products in the label space, in turn, it is independent from the representation of the output items and can be applied in any complex structural learning problem, e.g. on graphs. A suitable candidate for $\widetilde{\mathcal{Y}}$ could be the training set.

5.5 One-class SVM interpretation

Let us reformulate the inner-product occurring in the constraints whilst the bias term being dropped

$$\begin{aligned}
\langle \boldsymbol{\psi}(\mathbf{y}_i), \mathbf{W}\boldsymbol{\phi}(\mathbf{x}_i)\rangle_{\mathcal{H}_\psi} &= \mathbf{tr}\big(\boldsymbol{\psi}(\mathbf{y}_i)'\mathbf{W}\boldsymbol{\phi}(\mathbf{x}_i)\big)\\
&= \mathbf{tr}\big(\mathbf{W}\boldsymbol{\phi}(\mathbf{x}_i)\boldsymbol{\psi}(\mathbf{y}_i)'\big) = \langle \mathbf{W}, \big[\boldsymbol{\psi}(\mathbf{y}_i) \otimes \boldsymbol{\phi}(\mathbf{x}_i)\big]\rangle_{\mathcal{H}_\psi \otimes \mathcal{H}_\phi}
\end{aligned}$$

thus, we have a one-class SVM problem living in the tensor product space of the feature and the label spaces, where $\otimes$ denotes the tensor product.

One can extend the range of applications by using not only tensor product but more general relationship between the output and input items, i.e.,

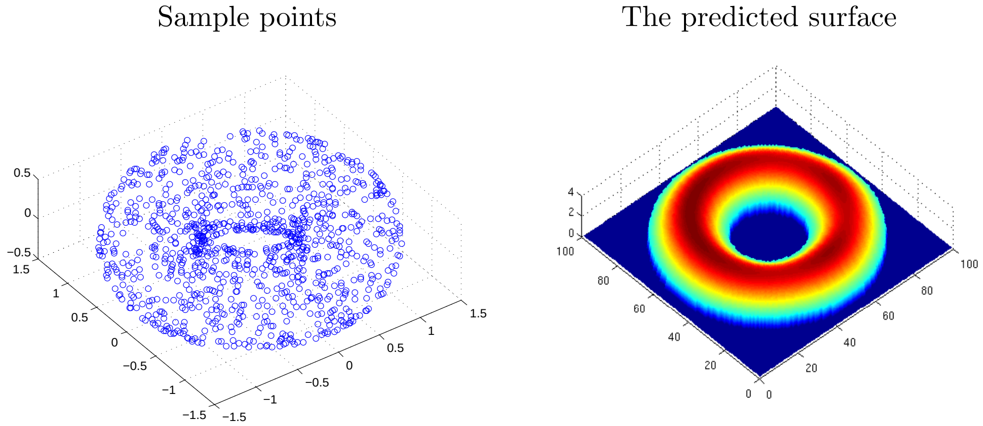
$$\langle \mathbf{W}, \boldsymbol{\Psi}(\mathbf{y}_i, \mathbf{x}_i)\rangle_{\mathcal{H}_W},\ \ \boldsymbol{\Psi} : \mathcal{H}_\psi \times \mathcal{H}_\phi \to \mathcal{H}_W.$$

If $\mathbf{dim}(\mathcal{H}_W) > \mathbf{dim}(\mathcal{H}_\psi) + \mathbf{dim}(\mathcal{H}_\phi)$ then the support of the distribution of one-class sample items is restricted on a manifold in $\mathcal{H}_W$. Further details of the extensions beyond the tensor product can be found in [12].

6 Preliminary results on shape estimation

On Figures 2 and 3 some preliminary results are presented. The first figure shows the sample of points of a torus, an object with hole, and the predicted surface learned of those sample points. The number of smaple point is equal to 200. To these point the corresponding normal vectors are computed and used in the prediction. The points are randomly and uniformly subsampled from the parametric representions of the torus.

The second figure demonstrates a complex object which consists of parts with significantly different geometries. The sample points of the shape is



Sample points

The predicted surface

Figure 2: Learning the shape of a torus, an object with hole, of randomly, uniformly sampled surface points

provided by the Microsoft Kinect device. To those point the normal vectors are estimated by the Point Cloud Library, an open source package, see details [6]. Within Figure 3 on the first image the points and the Support Vectors are shown, the second image additionally presents the confidence region around the point cloud.

Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 (Specific Programme Cooperation, Theme 3, Information and Communication Technologies) under grant agreement no. 270273, Xperience.

A Use of operators in the derivation of the kernels

When the kernels are derived we intensively exploiting the following rules connecting vectors of different vector spaces.

From two vectors of two distinct Hilbert spaces, $u_\alpha \in \mathcal{H}_\alpha$ and $u_\beta \in \mathcal{H}_\beta$ we can create an operator

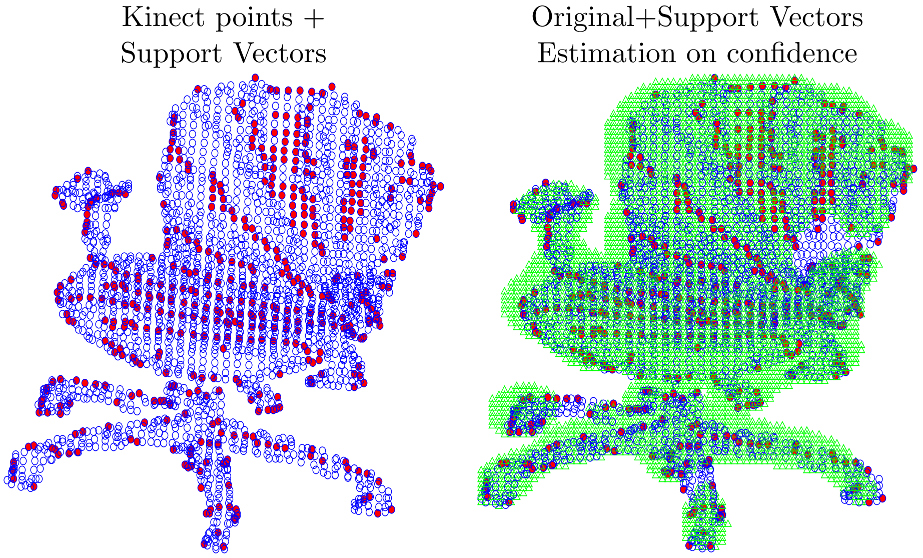
$$[u_\alpha \otimes u_\beta] : \mathcal{H}_\beta \rightarrow \mathcal{H}_\alpha, \tag{52}$$

which action on a vector v_β of $\mathcal{H}_\beta$ is defined by

$$[u_\alpha \otimes u_\beta] v_\beta \overset{def}{=} \langle u_\beta, v_\beta \rangle \, u_\alpha. \tag{53}$$

The conjugate of this operator is defined and denoted by

$$[u_\alpha \otimes u_\beta]^* \overset{def}{=} [u_\beta \otimes u_\alpha], \tag{54}$$



Kinect points +
Support Vectors

Original+Support Vectors
Estimation on confidence

Figure 3: Learning the surface of an armchair from a Kinect provided point set. On the left the sample points(blue) and the Support Vectors(red) are presented, on the right to those points on the left the confidence region is added(green)

which maps $\mathcal{H}_\alpha$ into $\mathcal{H}_\beta$.

The product of two operators

$$[u_\alpha \otimes u_\beta][v_\beta \otimes v_\gamma] : \mathcal{H}_\gamma \to \mathcal{H}_\alpha, \tag{55}$$

where

$$\begin{array}{l} [u_\alpha \otimes u_\beta] : \mathcal{H}_\beta \to \mathcal{H}_\alpha \\ [v_\beta \otimes v_\gamma] : \mathcal{H}_\gamma \to \mathcal{H}_\beta \end{array} \tag{56}$$

is defined as

$$[u_\alpha \otimes u_\beta][v_\beta \otimes v_\gamma]w_\gamma \overset{def}{=} \langle u_\beta, v_\beta \rangle \langle v_\gamma, w_\gamma \rangle u_\alpha \tag{57}$$

References

[1] K. Astikainen, L. Holm, E. Pitkanen, J. Rousu, and S. Szedmak. Reaction kernels, structured output prediction approaches for novel enzyme function. In *Conference on Bioinformatics 2010, Valencia*. 2010. Best Paper Award.

[2] D.P. Bertsekas. *Nonlinear Programming.* Athena Scienctific, second edition edition, 1999.

[3] J.M. Lee. *Introduction to Smooth Manifolds*, volume 218 of *Graduate Texts in Mathematics*. Springer, 2003.

[4] M. Popović, G. Kootstra, J. A. Jørgensen, D. Kragic, and N. Krüger. Grasping unknown objects using an early cognitive vision system for general scene understanding. In *2011 IEEE*. 2011.

[5] J. Rousu, C. Saunders, S. Szedmak, and J. Shawe-Taylor. Effcient algorithms for maxmargin structured classification. In *Predicting Structured Data*, pages 105–129. 2007.

[6] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[7] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. Smola, and R. Williamson. Estimating the support of a high dimensional distribution. *Neural Computation*, 13(7):1443 –1472, 2001.

[8] F. Steinke, M. Hein, J. Peters, and B. Schölkopf. Manifold-valued thinplate splines with applications in computer graphics. *Computer Graphics Forum*, 27(2):437–448, 2008.

[9] F. Steinke, B. Schölkopf, and V. Blanz. Support vector machines for 3d shape processing. *Computer Graphics Forum*, 24(3), EUROGRAPHICS 2005):285–294, 2005.

[10] S. Szedmak and Z. Hussain. A universal machine learning optimization framework for arbitrary outputs. 2009. http://eprints.pascalnetwork.org.

[11] S. Szedmak, Y. Ni, and S. R. Gunn. Maximum margin learning with incomplete data: Learning networks instead of tables. *Journal of Machine Learning Research, Proceedings*, 11, Workshop on Applications of Pattern Analysis:96–102, 2010. jmlr.csail.mit.edu/proceedings/papers/v11/szedmak10a/szedmak10a.pdf.

[12] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research (JMLR)*, 6(Sep):1453–1484, 2005.

Probabilistic object models for pose estimation in 2D images

Damien Teney[1] and Justus Piater[2]

[1] University of Liège, Belgium
Damien.Teney@ULg.ac.at
[2] University of Innsbruck, Austria
Justus.Piater@UIBK.ac.at

Abstract. We present a novel way of performing pose estimation of known objects in 2D images. We follow a probabilistic approach for modeling objects and representing the observations. These object models are suited to various types of observable visual features, and are demonstrated here with edge segments. Even imperfect models, learned from single stereo views of objects, can be used to infer the maximum-likelihood pose of the object in a novel scene, using a Metropolis-Hastings MCMC algorithm, given a single, calibrated 2D view of the scene. The probabilistic approach does not require explicit model-to-scene correspondences, allowing the system to handle objects without individually-identifiable features. We demonstrate the suitability of these object models to pose estimation in 2D images through qualitative and quantitative evaluations, as we show that the pose of textureless objects can be recovered in scenes with clutter and occlusion.

1 Introduction

Estimating the 3D pose of a known object in a scene has many applications in different domains, such as robotic interaction and grasping [1,6,13], augmented reality [7,9,19] and the tracking of objects [11]. The observations of such a scene can sometimes be provided as a 3D reconstruction of the scene [4], e.g. through stereo vision [5]. However, in many scenarios, stereo reconstructions are unavailable or unreliable, due to resource limitations or to imaging conditions such as a lack of scene texture.

This paper addresses the use of a single, monocular image as the source of scene observations. Some methods in this context were proposed to make use of the appearance of the object as a whole [6,13,15]. These so-called *appearance-based* methods however suffer from the need of a large number of training views. The state-of-the-art methods in the domain rather rely on matching characteristic, local features between the observations of the scene and a stored, 3D model of the object [1,7,17]. This approach, although efficient with textured objects or otherwise matchable features, would fail when considering non-textured objects, or visual features that cannot be as precisely located as the texture patches or geometric features used in the classical methods. Hsiao et al.'s method [8] seeks to better handle multiple possible correspondences between the model and scene features, but still requires a large fraction of exact matches to work efficiently.

The proposed method follows a similar approach to the aforementioned references for modeling the object as a 3D set of observable features, but it is different in the sense that few assumptions are made about the type of features used, and in that it does not rely on establishing specific matches between features of the model and features of the observed scene. For this purpose, we represent both the object model and the 2D observations of a scene as probabilistic distributions of visual features. The model is built from 3D observations that can be provided by any external, independent system. One of the main interests of the proposed method, in addition to the genericity of the underlying principles, is its ability to effectively handle non-textured objects. The general method itself does not make particular assumptions about the type of features used, except that they must have a given, although not necessarily exact, position in space, and they must be potentially observable in a 2D view of the object.

In order to demonstrate the capabilities of the proposed method at handling textureless objects, we apply it to the use of local edge segments as observations. Practically, such features cannot be precisely and reliably observed in 2D images, e.g., due the ambiguity arising from multiple close edges, 3D geometry such as rounded edges, or depth discontinuities that change with the point of view. Such problems motivate the probabilistic approach used to represent the scene observations.

The 3D observations used to build the model are provided by an external system that performs stereopsis on a single pair of images. Such a model can thus be quickly and automatically learned, at the expense of imprecision and imperfections in the model. This again motivates the use of a probabilistic distribution of features as the object model. Other *model-based* methods proposed in the literature have used rigid learned [7,17] or preprogrammed (CAD) models [9,19], but such CAD models are, in general, not available. Our approach for object modeling is more similar to the work of Detry et al. [5], where an object is modeled as a set of parts, themselves defined as probability distribution of smaller visual features. The main contribution of this paper is the extension of those principles to the use of 2D observations.

The representations of the object model and of the scene observations that we just introduced can then be used to perform pose estimation in monocular images, using an inference mechanism. Algorithms such as belief propagation [5] and Metropolis-Hastings MCMC methods [4] were proposed in the literature to solve similar problems, and we adapt the algorithm presented in that last reference to our specific type of model and observations.

Finally, our method provides a rigorous framework for integrating evidence from multiple views, yielding increased accuracy with only a linear increase of computation time with respect to the number of views. Using several views of a scene is implicitly accomplished when using a stereo pair of images, together with a method operating on 3D observations [5]. However, our approach does not seek matches between the two images, as stereopsis does, and can thus handle arbitrarily wide baselines. Other methods for handling multiple views with a 2D method have been proposed [2,14]. In these methods however, the underlying process relies on the matching of characteristic features.

2 Object Model

Our object model is an extension of earlier work [4]. For completeness and clarity, the upcoming sections include essential background following this source.

2.1 General form

We use a 3D model that allows us to represent a probabilistic distribution of 3D features that compose the model. These features must be characterized by a localization in the 3D space, and can further be characterized by other observable characteristics, such as an orientation or an appearance descriptor. The model of an object is built using a set

$$M = \left\{ \left(\lambda^{\ell}, \alpha^{\ell} \right) \right\}_{\ell \in [1,n]} \tag{1}$$

of features, where $\lambda^{\ell} \in \mathbb{R}^3$ represents the location of a feature, and $\alpha^{\ell} \in \mathcal{A}$ is a (possibly zero-element) vector of its other characteristics from a predefined appearance space $\mathcal{A}$. When learning an object model, the set of features M is decomposed into q distinct subsets M_i, with $i \in [1, q]$, which correspond ideally to the different parts of the object. This step allows the pose estimation algorithm presented below to give equal importance to each of the parts, therefore avoiding distinctive but small parts being overwhelmed by larger sections of the object. The procedure used to identify such parts is detailed in [4].

Our method relies on a continuous probability distribution of 3D features to represent the model. Such a distribution can be built using Kernel Density Estimation (KDE), directly using the features of M_i as supporting particles [5,18]. To each feature of M_i is assigned a kernel function, the normalized sum of which yields a probability density function $\psi_i(x)$ defined on $\mathbb{R}^3 \times \mathcal{A}$. The kernels assigned to the features of M_i will depend on the type of these features.

Reusing the distribution of 3D features of part i, ψ_i, and considering an intrinsically calibrated camera, we now define $\psi'_{i,w}$ as the 2D projection onto the image plane of that distribution set into pose w, with $w \in SE(3)$, the group of 3D poses. Such a distribution is defined on the 2D appearance space, which corresponds to $\mathbb{R}^2 \times \mathcal{B}$, where $\mathcal{B}$ is the projected equivalent of $\mathcal{A}$. For example, if $\mathcal{A}$ is the space of 3D orientations, $\mathcal{B}$ would be the space of 2D orientations observable on an image. Similarly, if $\mathcal{A}$ is a projection-independent appearance space of 3D features, $\mathcal{B}$ would be the simple appearance space of direct 2D observations of such features.

Practically, $\psi'_{i,w}$ can be obtained by setting the features of M_i into pose w, and projecting them onto the image plane (Fig. 1c). The resulting 2D features $\in \mathbb{R}^2 \times \mathcal{B}$ can, similarly to the 3D points, be used as particles to support a KDE on that space, using an equivalent projection of the kernels used in 3D.

2.2 Use of edge segments

This paper presents the particular application of the object model presented above to the use of local edge segments as visual features. Those features basically correspond to 3D oriented points, which are characterized, in addition to their localization in 3D, by an orientation along a line in 3D. Therefore, reusing the notations introduced above, the space $\mathcal{A}$, on which the elements α^ℓ are defined, corresponds to the half 2-sphere S^2_+, i.e. half of the space of 3D unit vectors. The kernels used to compose a 3D probability distribution ψ_i can then be decomposed into a position and an orientation part [5,18]. The first is chosen to be a Gaussian trivariate isotropic distribution, and the latter a von Mises-Fisher distribution on S^2_+. The bandwidth of the position kernel is then set to a fraction of the size of the object, whereas the bandwidth of the orientation kernel is set to a constant. The 2D equivalent of those distributions are obtained using classical projection equations. Fig. 2 depicts the correspondence between the 2D and 3D forms of a particle corresponding to an edge segment and its associated kernel.

The visual features used in our implementation are provided by the external Early Cognitive Vision (ECV) system of Krüger et al. [12,16]. This system extracts, from a given image, oriented edge features in 2D, but can also process a stereo pair of images to give 3D oriented edge features we use to build object models (Fig. 1b).

3 Scene observations

The observations we can make of a scene are modeled as a probability distribution in a similar way to the model. The observations are given as a set

$$O = \left\{ \left(\delta^\ell, \beta^\ell \right) \right\}_{\ell \in [1,m]} \tag{2}$$

of features, where $\delta^\ell \in \mathbb{R}^2$ is the position of the feature on the image plane, and $\beta^\ell \in \mathcal{B}$ are its observable characteristics. These characteristics must obviously be a projected equivalent to those composing the object model. Here again, the features contained in O can directly be used as particles to support a continuous probability density, using KDE.

In the particular case of edge segments, the observations correspond to 2D oriented points (Fig. 1e). They are thus defined on $\mathbb{R}^2 \times \mathcal{B}$ with $\mathcal{B} = [0, \pi[$. As mentioned before, the uncertainty on the position and orientation of visual features like edge segments can arise from different sources, and no particular assumptions can thus be made on the shape of their probability distribution. The kernels used here are thus simple bivariate isotropic Gaussians for the position part, and a mixture of two antipodal von Mises distributions for the orientation part. The sum of those kernels, associated with each point of O, then yields a continuous probability density function $\phi(x)$ defined on $\mathbb{R}^2 \times [0, \pi[$ (Fig. 1f).

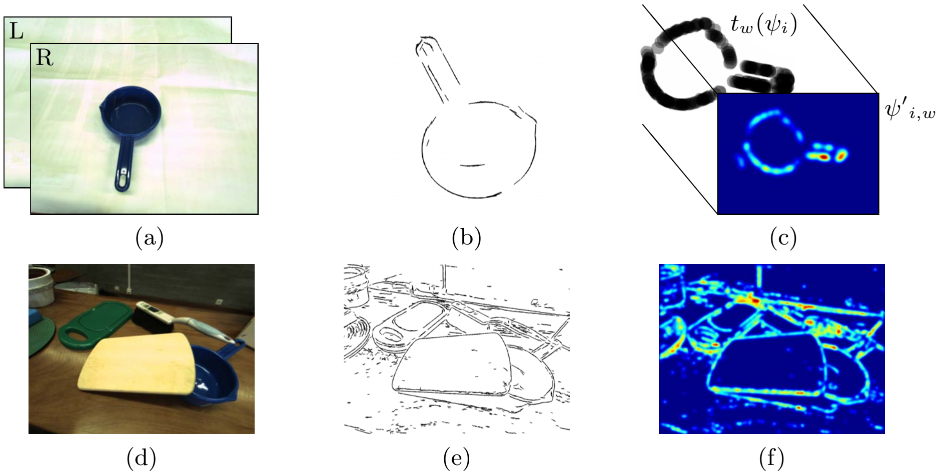


Fig. 1: Proposed method applied to edge segments (orientation of segments not represented). (a) Stereo images used to build object model; (b) 3D edge segments that compose the model; (c) probabilistic model (ψ_i) in pose w, spheres representing the position kernel (their size is set to one standard deviation), and its simulated projection in 2D ($\psi'_{i,w}$; blue and red represent resp. lowest and highest probability densities); (d) image of a scene; (e) 2D edge segments used as observations; (f) probabilistic representation of observations (ϕ).

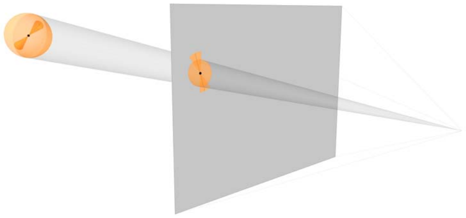


Fig. 2: Correspondence of 3D edge segment and associated kernel, with their 2D projection on image plane. Orange boundaries represent one standard deviation.

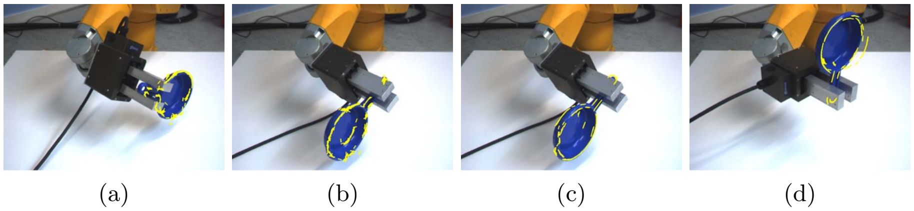


Fig. 3: Results of pose estimation; model features reprojected on input image. (a) Good result (close to ground truth); (b) good result; (c) same frame as (b) with incorrect result, orientation error of about 80°, even though the reprojection matches observations slightly better than (b); (d) incorrect result, insufficient observations extracted from pan bottom, and orientation error of about 180°.

4 Pose estimation

The object and observation models presented above allow us to estimate the pose of a known object in a cluttered scene. This process relies on the idea that the 2D, projected probability distribution of the 3D model defined above can be used as a “template” over the observations, so that one can easily measure the likelihood of a given pose.

Let us consider a known object, for which we have a model composed of q parts M_i ($i \in [1, q]$), which in turn define ψ_i and $\psi'_{i,w}$. On the other hand, we have a scene, defined by a set of observations O, leading to a probabilistic representation ϕ of that scene. We model the pose of the object in the scene with a random variable $W \in SE(3)$. The distribution of object poses in the scene is then given by

$$p(w) \propto \prod_{i=1}^{q} m_i(w) \ , \tag{3}$$

with $m_i(w)$ being the cross-correlation of the scene observations $\phi(x)$ with the projection $\psi'_{i,w}$ of the ith part of the model transformed into pose w, that is,

$$m_i(w) = \int_{\mathbb{R}^2 \times \mathcal{B}} \psi'_{i,w}(x) \, \phi(x) \, \mathrm{d}x \ . \tag{4}$$

Computing the maximum-likelihood object pose $\arg\max_w p(w)$, although analytically intractable, can be approximated using Monte Carlo methods. We extend the method proposed in [4], which computes the pose via simulated annealing on a Markov chain. The chain is defined with a mixture of local- and global-proposal Metropolis Hastings transition kernels. Simulated annealing does not guarantee convergence to the global maximum of $p(w)$, and we thus run several chains in parallel, and eventually select the best estimate. In practice, a strong prior is usually available concerning the distance between the camera and the object, e.g., as information on the scale at which the object can appear in an image. The global transition kernel can benefit from this prior to favor more likely proposals, and therefore drive the inference process more quickly towards the global optimum.

As mentioned above, the proposed method naturally extends to observations from v multiple views. We define $m_{i,j}(w)$ similarly to Eq. 4 but relative to specific views j, $j = 1, \ldots, v$. Accounting for observations from all available views, Eq. 3 then becomes

$$p(w) \propto \prod_{j=1}^{v} \prod_{i=1}^{q} m_{i,j}(w) \ , \tag{5}$$

which is handled by the inference process similarly to the single-view case.

5 Evaluation

This sections presents the applicability of the proposed method for estimating the pose of objects on two publicly available datasets [3,10].

5.1 Experimental setup

In this work, each model is built from one manually segmented stereo view of the object (such as Fig. 1a). The models used here are typically composed of between 1 and 4 parts, containing around 300 to 500 observations in total. Pose estimation is performed on single 1280 × 960 images taken with an intrinsically calibrated camera. The number of parallel inference processes (see Section 4) is set to 16. On a typical 8-core desktop computer, the pose estimation process on a single view typically takes about 20 to 30 seconds. Also, as proposed in Section 4 and detailed below, a crude estimate of the distance between the camera and the object is given as an input to the system.

The ECV observations we use (see Section 2.2) can be characterized with an appearance descriptor composed of the two colors found on the sides of the edge. This appearance information does not enter into the inference procedure. However, in the following experiments we use it to discard those scene observations whose colors do not match any of the model features. This step, although not mandatory, helps the pose estimation process to converge more quickly to the globally best result by limiting the number of local optima.

5.2 Rotating object

We first evaluated our method on a sequence showing a plastic pan undergoing a rotation of 360° in the gripper of a robotic arm [10]. The ground truth motion of the object in the 36 frames of the sequence is thus known. The estimate of the distance to the object, given as input to the system, is the same for the whole sequence, and is a rough estimate of the distance between the gripper and the camera (about 700 mm). Let us note that, for some images of the sequence, this estimate is actually quite different from the exact object-camera distance, since the object is not rotating exactly around its center.

This publicly available dataset is composed of stereo images, and we used the frame corresponding to a rotation of 50° to learn the model, as it gives a good overall view of the object. Four types of experiments were then performed (Fig. 4). First, the pose of the object was estimated in each frame of the sequence, using one single view. One can observe that correct pose estimates can mostly be made close to the viewpoint used for learning the model (Fig. 4). A number of results have an orientation error of almost 180°, which correspond to a special case (Fig. 3d) that can be explained by the flat and almost symmetrical object we consider. Indeed, if very few observations are extracted from the bottom of the pan, only the handle and the top rim of the object can be matched to the image. Another large number of incorrect pose estimates have orientation errors of 70–110°; most of them correspond to ambiguities inherent to a 2D projection, as illustrated on Fig. 3b–c. Similarly, most of the translation errors occur along the camera-object axis, as an inherent limitation of 2D observations. The percentage of correct pose estimates, defined by orientation and translation errors of less than 10° and 30 mm resp., and evaluated over the whole sequence, is only 20%. Second, the same experiment is performed using two views. Some of the ambiguities can then be resolved, and this percentage rises to 60%. This result can be compared to the evaluation of Detry et al. [5] on a similar sequence, which achieved a score of only 40–50%. We stress that the latter method relied on 3D observations computed from stereo, whereas our method uses one or more 2D images directly, and is not limited to short-baseline stereo pairs.

Finally, we used our framework to track the pose of the object over the whole sequence, using one and two views, respectively. The pose is initialized with ground truth information for the first frame, and is then tracked from one frame to the next, using the same process as outlined in Section 4, but without the use of global proposals in the chain, and thus limiting the inference process to a local search. These experiments yield very good results (see Fig. 4), the remaining error being mostly due to the limitations of the model, learned from a single view of the object.

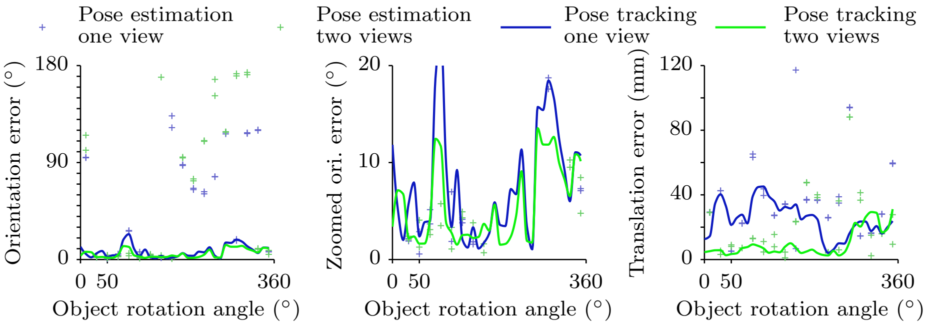


Fig. 4: Results of the "rotating object" sequence. For pose estimation, one marker represents one run of the algorithm (the same number of runs are executed for each frame). For pose tracking, the lines represent means over multiple runs.

5.3 Cluttered scenes

We evaluated the robustness of our method to clutter and occlusions by computing the pose of various objects in several cluttered scenes [3], using a single input image. The estimate of the distance to the objects, used as input, is the same for all scenes and objects, and roughly corresponds to the distance between the camera and the table on which the objects are placed (about 370 mm). Here again, this is an only crude estimate, as the actual distance to the objects varies from 200 to 600 mm.

Several of these scenes are presented in Fig. 5, with object models superimposed in the estimated pose. Sometimes, insufficient observations are extracted from the image, and the pose cannot be recovered (e.g. second row, last image). However, the reprojection error achieved by our algorithm is clearly low in most cases; the models generally appear in close-to-correct poses. A perfect match between the reprojected model and the observations is not always possible, which is a limitation of the sparse observations and object models we use. Small differences in the reprojection on the image plane may then correspond to large errors in the actual 3D pose recovered. Most of these errors can be greatly reduced by using additional views of the scene, which is easily done with our method.

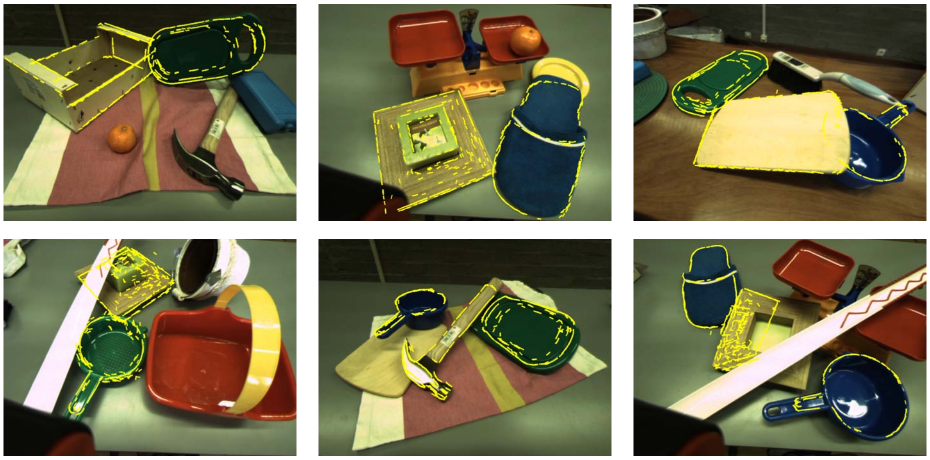


Fig. 5: Results of pose estimation (using a single view), with model features reprojected onto the input image. Most remaining errors are a limitation of the simple object models used, each learned from a single stereo pair.

6 Conclusions

We presented a generic method for 3D pose estimation of objects in 2D images, using a probabilistic scheme for representing object models and observations. This allows the method to handle various types of observations, including features that cannot be matched individually; here we use local edge segments. Using these principles, we showed how to use Metropolis-Hastings MCMC to infer the maximum-likelihood pose of a known object in a novel scene, using a single 2D view of that scene. The probabilistic approach makes the pose estimation process possible without establishing explicit model-to-scene correspondences, as opposed to existing state-of-the-art methods. Together with the use of edge segments as observations, the method allows us to effectively handle non-textured objects. Further, the method extends to the use of multiple views, providing a rigorous framework for integrating evidence from multiple viewpoints of a scene, yielding increased accuracy with only a linear increase of computation time with respect to the number of views. We validated the proposed approach on two publicly-available datasets. One dataset allowed quantitative evaluation; the result of an experiment was compared to the results of an existing method, and showed an advantage in performance for our method. The pose estimation process was also evaluated with success on scenes with clutter and occlusion. Future work will extend the current implementation to the use of other visual features, thereby extending the types of objects that can be handled.

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 (Specific Programme Cooperation, Theme 3, Information and Communication Technologies) under grant agreement no. 270273, Xperience. Damien Teney is supported by a research fellowship of the Belgian National Fund for Scientific Research.

References

1. Collet, A., Berenson, D., Srinivasa, S., Ferguson, D.: Object recognition and full pose registration from a single image for robotic manipulation. In: ICRA (2009)
2. Collet, A., Srinivasa, S.S.: Efficient multi-view object recognition and full pose estimation. In: ICRA. pp. 2050–2055 (2010)
3. Detry, R.: A probabilistic framework for 3D visual object representation: Experimental data (2009), `http://intelsig.org/publications/Detry-2009-PAMI/`
4. Detry, R., Piater, J.: Continuous surface-point distributions for 3D object pose estimation and recognition. In: ACCV (2010)
5. Detry, R., Pugeault, N., Piater, J.: A probabilistic framework for 3D visual object representation. IEEE Trans. PAMI 31(10), 1790–1803 (2009)
6. Ekvall, S., Hoffmann, F., Kragic, D.: Object recognition and pose estimation for robotic manipulation using color cooccurrence histograms. In: IROS (2003)
7. Gordon, I., Lowe, D.G.: What and where: 3D object recognition with accurate pose. In: Toward Category-Level Object Recognition. pp. 67–82 (2006)
8. Hsiao, E., Collet, A., Hebert, M.: Making specific features less discriminative to improve point-based 3D object recognition. In: CVPR. pp. 2653–2660 (2010)
9. Klein, G., Drummond, T.: Robust visual tracking for non-instrumented augmented reality. In: ISMAR. pp. 113–122. Tokyo (October 2003)
10. Kraft, D., Krüger, N.: Object sequences (2009), `http://www.mip.sdu.dk/covig/sequences.html`
11. Kragic, D., Miller, A.T., Allen, P.K.: Real-time tracking meets online grasp planning. In: ICRA. pp. 2460–2465 (2001)
12. Krüger, N., Wörgötter, F.: Multi-modal primitives as functional models of hypercolumns and their use for contextual integration. In: Gregorio, M.D., Maio, V.D., Frucci, M., Musio, C. (eds.) BVAI. Lecture Notes in Computer Science, vol. 3704, pp. 157–166. Springer (2005)
13. Mittrapiyanuruk, P., DeSouza, G.N., Kak, A.C.: Calculating the 3D pose of rigid objects using active appearance models. In: ICRA. pp. 5147–5152 (2004)
14. Pless, R.: Using many cameras as one. In: CVPR (2). pp. 587–593 (2003)
15. Pope, A.R., Lowe, D.G.: Probabilistic models of appearance for 3D object recognition (2000)
16. Pugeault, N.: Early Cognitive Vision: Feedback Mechanisms for the Disambiguation of Early Visual Representation. Vdm Verlag Dr. Müller (2008)
17. Rothganger, F., Lazebnik, S., Schmid, C., Ponce, J.: 3D object modeling and recognition using local affine-invariant image descriptors and multi-view spatial constraints. Int. J. Comput. Vision 66(3), 231–259 (2006)
18. Sudderth, E.B.: Graphical models for visual object recognition and tracking. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, USA (2006)
19. Vacchetti, L., Lepetit, V., Fua, P.: Stable real-time 3D tracking using online and offline information. IEEE Trans. PAMI 26(10), 1385–1391 (2004)