Robots that Validate Learned Perceptual Models

Ulrich Klank, Lorenz Mösenlechner, Alexis Maldonado and Michael Beetz
Department of Informatics, Technische Universität München
{klank,moesenle,maldonad,beetz}@cs.tum.edu

***Abstract*— Service robots that should operate autonomously need to perform actions reliably, and be able to adapt to their changing environment using learning mechanisms. Optimally, robots should learn continuously but this approach often suffers from problems like over-fitting, drifting or dealing with incomplete data.**

In this paper, we propose a method to automatically validate autonomously acquired perception models. These perception models are used to localize objects in the environment with the intention of manipulating them with the robot.

Our approach verifies the learned perception models by moving the robot, trying to re-detect an object and then to grasp it. From observable failures of these actions and high-level loop-closures to validate the eventual success, we can derive certain qualities of our models and our environment. We evaluate our approach by using two different detection algorithms, one using 2D RGB data and one using 3D point clouds. We show that our system is able to improve the perception performance significantly by learning which of the models is better in a certain situation and a specific context. We show how additional validation allows for successful continuous learning. The strictest precondition for learning such perceptual models is correct segmentation of objects which is evaluated in a second experiment.

I. Introduction

To enable a robot to perform complex actions such as cooking meals and cleaning up skillfully and reliably in a highly dynamic and changing environment such as a human household, adaptation to the environment is essential. In this paper, we discuss the special problem of encountering unknown objects and learning perception models to recognize these objects. However, most learning methods suffer in some way from over-fitting and over-specialization or on the other hand from a lack of expressiveness to encode required facts. Furthermore, self-supervised learning schemes tend to diverge over a longer continuous operation towards unwanted or unexpected behavior.

In this paper, we show that those general problems related to learning may be overcome by exploiting redundancy and self-validation which is present inside an environment the robot can interact with. More specifically, we present our system setup around the perception system `Cognitive Perception` (CoP) that interacts with the environment to validate object models and derive the quality of a learned model. In essence, the robot verifies predictions it made based on its current belief state. This allows to assess the quality of a learned model, which we will refer to as *high-level loop-closure*. We will show in our experiments two examples of predictive models and their verification: in the first experiment, the robot moves to detect a (re-detectable) object on the table from different perspectives. During this process, the objects on the table are assumed to be static. Second, a well segmented and rigid object will be moved by the robot taking it and placing it somewhere else. In the first experiment, we show the validity of a model by detecting an object from different view-points. Here, the loop-cosure happens by a successful re-detection of an object at the same place according to the odometry which indicates a valid model. In the second experiment we perform a pick-and-place action to verify the segmentation of an object. The loop-closing event is here the following: If the object has been segmented correctly we can pick it up and put it down at a different location and still re-detect the object at the now expected new position using the previously learned model. Failure indicates a poor model which should be abandoned.

We significantly improve the recognition performance of the system by allowing the robot to validate its models. This is implemented by using the outcome of actions as input for the feedback system of CoP. This learning process transparently integrates into the robot's high-level task execution system, i.e. learning is done without increasing the calculation times of object recognition and without adding offline computations. In our test scenario, the performance increase is measured by the capability to decide which of two learned perceptual models is better or if none of them is applicable in the current situation. One of the two models is based on the texture and silhouette, the other model on the 3D shape. Figure 1 shows an example of a scene where four objects cannot be distinguished by their shape, but significantly by their texture.

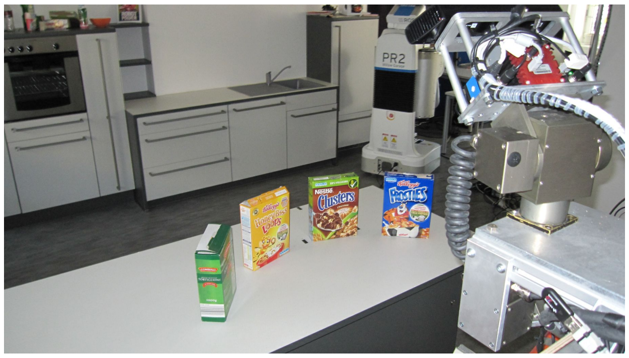


Fig. 1. To detect some objects, texture is more relevant than shape, for other objects, shape is more relevant.

The integration of the perception system in the robot's entire system is shown in Figure 2. The control of the robot has to be accurate and failure and error aware, i.e. all components need to be able to detect failures such as objects slipping out of the robot's gripper. This is necessary be able to perform high-level loop-closure. The entire system is considered as the contribution of this paper. It allows for continuous self validation of several aspects of its perception mechanism and enables self-supervised learning. Additionally, we introduce several minor improvements to the used perception mechanism and several interesting design patterns in the used software which are required for the system to work.

In the remainder of this paper we proceed as follows: After discussing related work, we will introduce the perception system in Section III and will give more background on the used methods. Section IV we describe the executive which is closing the high-level loop. It uses the perception system, controls the robot accordingly and provides feedback for the perception system. We will continue with the implementation of the feedback and the actions that are triggered by positive or negative feedback in Section V. Finally, Section VI, we will present the results of our experiments.

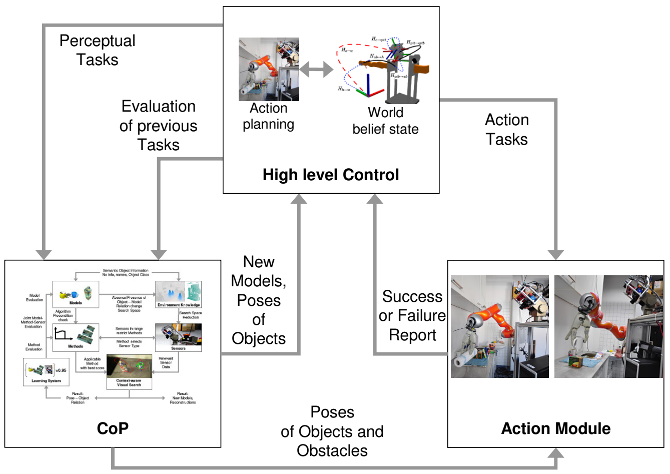

Perceptual Tasks
Evaluation of previous Tasks
Action planning
World belief state
High level Control
Action Tasks
New Models, Poses of Objects
Success or Failure Report
CoP
Action Module
Poses of Objects and Obstacles

Fig. 2. The integration of the executive with the perceptual system and the actuation system.

II. Related Work

Perceptual modeling of objects is a widely discussed problem and is often done semi-autonomously in order to achieve better performance. Examples for automatic approaches are Lai et al. [1] where the authors provide a strong RGB-D descriptor for segmented objects or the work presented in [2] which only expects that a robot has an object in its gripper in order to learn textured 3D-representations.

There are currently several challenges set up in the area of perception, which are interesting to compare with the problem we impose here. For instance, the "Solutions In Perception Challenge" was proposed and discussed in a recent IROS Workshop [3]. It requests the recognition of objects in cluttered scenes. In contrast to the presented approach here, the provided data of objects are assumed to be well segmented, rigid, textured and lambertian and views are available from all directions. Such data, fulfilling the above assumptions, cannot always be provided in an unstructured environment. [4] assumes segmentation and validation over multiple views, which results in nice 3D object representations, but unfortunately requiring a static world. A bit different is the approach proposed in [5] which derives parameters for predefined functional models that are applied on movable objects to extract basic properties. This approach requires a function to explain all possible effects and needs means to measure those effects, and both things are not intrinsically verifiable.

III. Perception System

The perception system CoP selects and applies different perception methods and manages sensors and perception models to detect and re-detect a big variety of different objects. A perceptual model (in the remainder of this article just model) is annotated information that can be used to segment, identify or localize types of objects. A simple example is a CAD model of a mug, which is used by an appropriate method to localize objects that show approximately the 3D shape described by the CAD model. A perception method is anything that makes use of such models or learns new perceptual models. The sensors are intuitively defined as any incoming data which is not annotated by a high-level system.

The basic data structure controlling the perception system CoP for the segmentation and other basic properties is a mapping of visual tasks onto a success statistics, wherein the visual task is represented as a combination of a method, a model, and a sensor configuration in space (i.e. its pose):

$$\textit{vis-task} \times \langle \textit{method,model,sensor} \rangle \longrightarrow \textit{success statistics}$$

where each method is applicable only to a subset of model types and specific sensor configurations. The statistics for these methods, models and sensor combinations then form the basis for selecting the task specific perception methods. The acquisition of the success statistics is supervised by the high-level control system of the robot that provides the feedback. This feedback contains information about whether or not a perceptual task resulted in a successful task execution. For example, the high-level controller might specify in the context of a pick-up task that the localization of an object was successful if the robot successfully grasped the object. Otherwise the success of the visual task is unknown since a grasp failure could have been caused by the hand or arm controller or the perception routine.

CoP essentially learns in two ways: first, given detected object candidates, it creates new perceptual models for objects and extends the set of applicable $\langle \textit{method,model,sensor} \rangle$ triples. Second, it learns the decision rules for selecting the appropriate method/model/sensor combination by continuously collecting the results of their application whenever they have been applied in the context of problem-solving. We approximate the quality of a model at a specific time instant based on the acquired feedback information for a model using the following formula:

$$p(M) = \frac{p_M + \sum_{\forall f \in F_{M,O}} f}{1 + \|F_{M,O}\|}$$

where p_M is the overall performance of the method M, which is initially given for all methods and $F_{M,O}$ being the list of all feedbacks that were reported regarding the method M for the object O, a feedback f is a value between 0.0 (fail) and 1.0 (success). The highest $p(M_i)$ decides on the method M_i to be chosen for the next run.

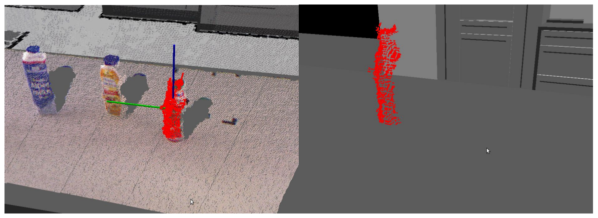
In order to learn a perceptual model for a new object a segmentation is necessary. By the segmentation, an object hypothesis is formulated. The better this first hypothesis is, the more valuable the later objects will be. All yet tested segmentation methods fail from time to time, so any model has to be considered as potentially wrong.

For the evaluation of this paper, we used, as mentioned before, two methods to learn and localize objects. Even if CoP in its current implementation contains more methods, we will exemplary evaluate our system with only two methods. The exact description of the perception methods is only relevant for understanding and reproducing the experiments, though only the integration into the error-aware system is part of the contribution of this article.

Both of the methods have a fundamental property that allows easier evaluation: the localization works without previous segmentation of a scene, but may be applied to only parts of an image. The models of the objects are also robust enough to withstand sensor noise and are general enough to be applied to data from different sensors. Both models can be learned successfully from only one view of the object if the segmentation is correct. Additionally, both methods have a similar runtime in our setup, which is below half a second for the hardware setup we use.

A. Surface Model Extraction and Matching

Assuming the surface of the object is distinctive, we can extract from a segment of a depth image a model describing this surface using the relations of calculated normals on the surface. The relation of two identified points and their normals is already sufficient to estimate a pose of the object. By performing a voting in a new depth image on positions in space, the method presented in [6] relocates the template again in different images (see Figure 3). The pose winning the voting is refined with an ICP algorithm (Interactive closest point) between the model points and the new depth image. The voting is performed in a discretized 4D space representing candidate locations of the object, given a selected model point is at a certain position with a known normal. Such a point is called seed point and all other points of a sub-sampled version of the current scene are voting for a position with orientation in this space. After several seed points are tested, there are strong maxima for several of the seed points which correspond to the best match in the image. This method will find a present object with a high probability, given the sampling in the scene is dense enough and it has a unique shape with respect to the background and clutter.


(a) A point cloud segment used for building a surface model. (b) A point cloud segment used for building a surface model.

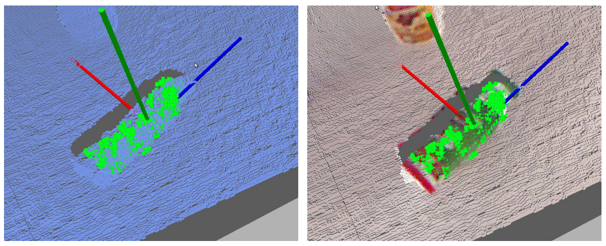

(c) A match and the transformed segment at a new position. (d) The same match with a colored point cloud.

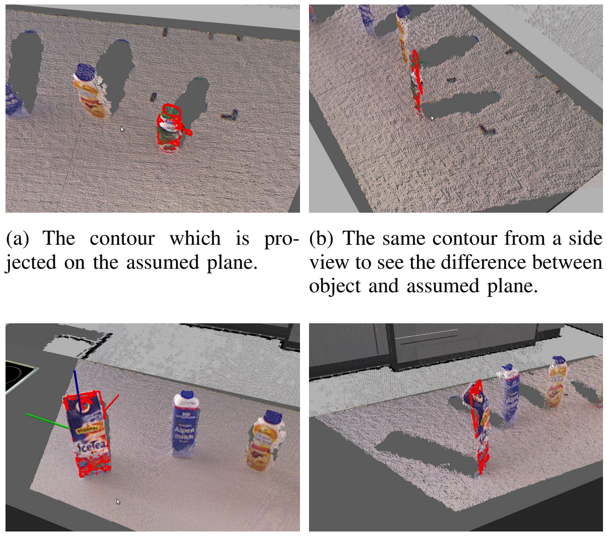
Fig. 3. Surface model for localization of Objects.

The execution time of this method scales with the 3D space that has to be searched, and comes to near real-time performance in our setup if we reduce the search space in the depth image to the current working volume without known furniture.

B. Planar Shape Model Extraction and Matching

If we assume partial planarity of an object with sufficient size, we can learn a shape model of the object. It is helpful if the object is textured, but a prominent outer contour may already be sufficient for a good recall. By saving a 3D position of the estimated planar substructure and the image region's barycenter, a learned template can be re-localized in 3D (see Figure 4). For extracting the main planar substructure we just calculate the SVD (Singular value decomposition) of the segmented 3D points, and use the cross product of the first two Eigenvectors as a plane normal through the object's center. This is done unless the normal direction is too different from the viewing direction, which will lead to the usage of a fall back solution taking the second most prominent plane, assuming that a steeper view will make the learned model worse.

A shape model generated from an earlier view of an object is used to re-localize the object. With such a template we can calculate a homography estimating the best 2D-2D hypothesis between the training image and the final image by using the method presented in [7]. Such a localization has a relatively high residual error in the viewing direction of the relevant sensor which is reduced by applying the method to a second camera if available. If the results are consistent up to a certain degree, the mean position and rotation of the two results is used.


(a) The contour which is projected on the assumed plane. (b) The same contour from a side view to see the difference between object and assumed plane.

(c) A contour for another object, frontal view. (d) A contour for another object, side view.

Fig. 4. Planar Shape model for localization of Objects.

IV. Executive

To close the high-level loop between perception and manipulation, we use an executive that allows for reasoning about plans and provides powerful mechanisms to react on different errors under various contexts. The executive is central component for generating feedback information used by the lower level parts of the system in a flexible and general way and to monitor continuous learning.

The executive is based on the CRAM Plan Language [8] which is a rewrite of Drew McDermott's RPL [9]. It provides a rich set of expressions that allow for the specification of highly parallel, synchronized, flexible and transparent plans. Failure handling is achieved by an extensive, exception based mechanism that also works across the boundaries of multiple processes. More specifically, the language provides constructs for parallel and sequential evaluation with various failure semantics. Examples include *par* which executes program instructions in parallel and terminates after all of them terminated or *pursue* which executes instructions in parallel and terminates when one of the sub-forms terminates.

Programs in our executive cannot only be executed but also reasoned about [10]. This is achieved by providing annotations for all higher-level actions such as picking up an object, putting it down, perceiving an object or moving the arm to a specific position. While executing a plan, an execution trace is generated that contains all information that is required to completely reconstruct the course of actions, the beliefs of the robot during action execution (i.e. the assignment of local and global variables) and the status of control routines at any point in time (e.g. failed, succeeded, etc.). Besides allowing us to infer what the robot did, when and why after a plan has been executed, this also allows us to directly infer the current context of actions *during* plan execution. This provides valuable context information about where a failure occurred and how it has been handled.

Causes for failures are manifold. The robot might fail to navigate to a specific location because it is unreachable. Manipulation might fail because the object cannot be reached which might have several reasons: the object might be too far away, the location of an object detected by perception might be inaccurate or the self-localization of the robot drifts too much and the robot misses the object while trying to grasp it. Perception might fail, too. Our system requires the lower-level routines such as navigation, manipulation or perception to detect and signal errors such as "object not detected in the gripper". These errors are handled locally by, for instance, changing the location of the robot, re-perceiving the object and retrying the action.

While executing an action, the system asserts and retracts information about the current context, currently executed actions and their outcome, results and errors in a knowledge base. There, a feedback mechanism that infers specific failure conditions can hook in. If a specific combination of execution context, failures and the corresponding parameterization match a failure pattern, a feedback message for the perception system is generated, containing all information that is required to evaluate the perception action involved in the error.

V. Perception Action Loop

In order to generate feedback in the executive while performing a task back to a perception system, we have to consider several important facts: first, we need a communication infrastructure that allows to pass detailed information about what happened and to which action of the perception system this information refers to. Second, any feedback will only help when it is consistent with the success or failure of the perception system itself.

We solved the communication by assigning a unique ID to each of the tasks which are performed by the perception system. In the perception system the connections of performed methods and models with this ID are memorized. In case of a feedback, the combination of method and model are evaluated accordingly, as well as any earlier action leading to the creation of the model.

Much harder to meet is the second condition: given the success of a plan or an error during execution, the direct conclusion that also the perception routines succeeded or failed is invalid. Therefore, the executive needs a deep insight in preconditions and effects of executed plans in order to solve that task. As an example we want to explain the details of the task we used to evaluate our system, a pick-and-place task.

A. Pick-and-Place Setup

In short, our plan performs only the following standard pick-and-place task:

1) Segment the scene
2) Learn models for all objects
3) Try to grasp one of them
4) Put down the object at a different location

5) Validate the predicted position with the learned models

More interesting are the possibilities to detect and recover from errors: If the models are learned and one of the objects is selected, for this object, the model can be preliminary validated by moving the robot in order to grasp the object and using the new model to localize the object again before grasping it. If this fails, the robot will retry which might already cause the use of another perceptual model that is also applicable in the current context. While grasping, the hand's torque parameters are observed to detect any unpredicted collisions which may give information about unseen objects or an inaccurate position estimation of the target. This information might already indicate that the robot has moved the object and in case nothing is grasped the object has to be localized again. Whether the robot grasped something is decided based on the joint angles of the hand after the grasp. When they indicate a fully closed hand instead of a hand holding an object, the grasp is considered a failure. Please see [11] for more details of the used grasping system.

If the object is put down to an empty spot on the table, the near environment of the put-down location is scanned for the object again. If it appears to be close enough to the predicted position, the entire process is considered successful. In most cases, the model is even robust enough to localize the object if it was not put down carefully enough to stand upright, i.e. if it flipped over after the put-down action. This situation may indicate a failure in the grasping, but is no indicator for the quality of the perception at this point.

B. Feedback Generation

The high-level control program executes the simple pick-and-place plan described above and observes the different errors that may occur. For instance, if the object is not reachable, the vision system is most probable not responsible for this problem but the executive's choice of a location to stand for reaching the object could have been bad. On the other hand, if the manipulation system does not detect the object in the gripper, the perceived object location was probably bad and the perception system receives a negative evaluation value. Other errors that occur include that the object could not be re-detected which indicates that the learned object model does not fit under the current environment configuration and location of the robot. Such errors are handled locally by moving the robot to a different location and trying to perceive and pick up the object from there. If the robot could successfully pick up the object, the system puts it down at a different location and then searches for it again at the put down location. Since the robots plans the position to put down the object, it has implicitly a qualified prediction about the expected location of the object. Therefore, the distance between a re-detection of the object and the expected location can be measured. If the object is close to the desired pose, the perception system receives a positive feedback, which is not only valid for the model used for identification but also for the segmentation that preceded the creation of the perceptual model. A re-detection of the object that is not close to the put-down pose indicates a false-positive and the currently learned model for that object receives a negative evaluation.

	Shape Model	Surface Model	Learning
Boxes	69.6%	55.0%	76.3%
Shapes	51.9%	62.5%	83.5%
Diverse	62.1%	60.0%	76.0%
All	67.9%	59.2 %	**78.2%**
||Measurements||	125	240	239

TABLE I

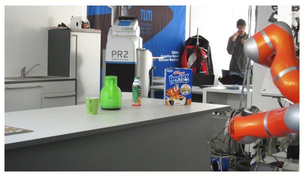
COMPARISON OF THE TWO METHODS AND THE LEARNING PROCESS OVER THE METHODS FOR DIFFERENT OBJECT SCENARIOS.

VI. RESULTS

In this section, we present the results of the two experiments mentioned earlier in the paper.

A. Evaluation of Model Selection

To get an intuition how often a one-shot learning model will succeed, and to visualize the differences of the models, we made the following experiment: We let the robot learn several objects in a scene from one view. Then the robot moves to a set of points inside a region of about 2 square meters. From all points the robot perceives the scene and tries to localize the previously learned objects in the scene. For this localization task, the search space is restricted to "on the table" and the assumption of the previously known location is dropped internally to prevent favoring of a similar location. In addition, two different scene setups were used: box-only scenes and diverse scenes as shown in Figure 5.


(a) One of the diverse scenes.

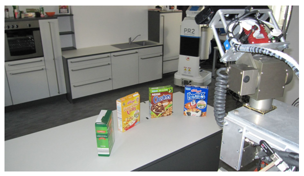

(b) One of the boxes-only scenes.

Fig. 5. Examples of scenes that were used to evaluate CoP's model selection.

By repeating this experiment, we extracted several key parameters of our system which are summarized in Table I: for most objects in our domain, i.e. a kitchen, the planar shape model performs slightly better than the surface based model. But this result is influenced by the selection of special subsets of objects. When only the boxes are taken into account, the shape model will be favored. On the other hand a majority of texture-less objects will favor the surface model. However, if the perception system is allowed to decide based on its experience for the learned models which model to use for localization, the performance is far better: Namely, we achieved 78.2% correct detection within the localization error of the robot compared to 59.0% and 68.0% for constantly favoring one method. This requires, of course, that after every trial, feedback about the success is provided to the system.

	Segmentable	Not Segmentable
Error in pickup phase	30%	50%
Error in put down phase	0%	10%
Model failed post move eval.	0%	20%
Plan Succeeded	**70%**	**20%**
|Trials|	30	10

TABLE II

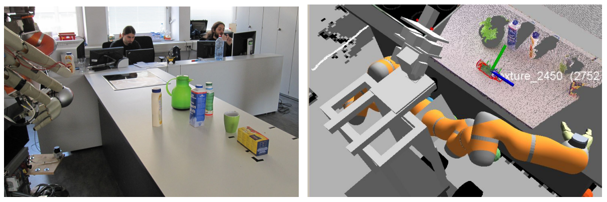
STATISTICS OF RESULT STATES OF THE PLAN FOR TEST SCENES.

The learning switched in average 2 - 3, maximally 5 times between the two models until the learning process converged for a static scene. E.g. in the `Boxes` scenario with several equal boxes and one distinct box, all but the distinct box converged to the shape model. This shows the high context awareness of the method, while never executing two methods at once in order to avoid additional execution time.

B. *Evaluation of the Pick-and-Place Plan*

Segmentation is a crucial precondition to learn a valid model for a new object candidate in nearly all current methods, including the two methods we used in this experiment. In order to validate the segmentation we perform the following test: the robot is only able to move an object away from its position when the previous segmentation was correct. Otherwise, it will only move parts of the object or it will fail when trying to pick up the object. We will apply the previously mentioned action (see Section V-A) to a scene which was set up with one of two modes:

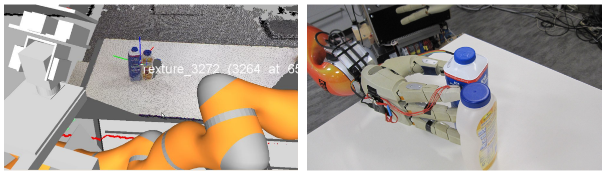
- easily segmentable, i.e the distance between objects is greater than five centimeters
- not segmentable in 3D space, i.e. two objects are (almost) touching each other



(a) A setup for the pick-and-place task. The objects are well segmentable.
(b) A view of the verification step, showing an object at a new position.

Fig. 6. Snapshots from the experiment, used to evaluate the segmentation and model building.

The resulting data from the experiment can be seen in Table II and can be interpreted as follows: in case of a successful segmentation, the objects could be always localized correctly using the learned models, which were used at least twice before. In both cases, in the segmentable scenes as well as in the not segmentable scenes our grasping system had some problems, in various cases caused by a bad model, which was not giving a sufficiently accurate localization. Examples for segmentable scenes can be found in Figure 6. In case of the not segmentable scenes we observed four times the trickier case when the robot moves part of the objects in the selected segment. Figure 7(a) shows such a segment and Figure 7(b) shows a grasp that was considered as successful while only lifting the left object. The corresponding model was then rejected due to low correlation of the placed object and the learned model. The bold numbers in Table II show cases which successfully generated information that could be passed to the perception system. In only two tests in which the plan was considered successful in the not segmentable scenes, did the information lead to a wrong believe state of the system about the quality of the model.



(a) Model learning step with failed segmentation.
(b) The subsequent grasp, that picked up one of the objects.

Fig. 7. The border case, when the grasping works even with wrongly segmented objects, has to be recognized.

VII. Discussion and Conclusions

We showed how we could improve the performance for a robotic perception system with a simple, self-supervised learning system implemented on a robot. The learning effect can be achieved independently of the underlying perception methods and could be applied to any newly acquired model that allows localization of an object. All feedback data was based on the capabilities of the robot to observe the outcome of certain actions in order to use this knowledge. Even if both experiments still show room for improvement in terms of robustness, we still see this as an interesting step towards autonomy for service robots in domestic environments. The results already show the feasibility of continuous self-supervised learning in complex environments with challenging tasks without risking diverging if the system exploits implicit knowledge gathered through monitoring actions and their outcome.

The manipulation system under the uncertainty of badly segmented and represented objects still has some problems that might be partially fixed by using more sensors to detect contact in the hand and better grasp planners. We also did not yet consider the calculation time to decide which perceptual model will be used. This could lead to the possibility of using combinations of methods, which could increase the robustness of the system further.

Acknowledgments

This work is partially supported by MVTec GmbH, München, *www.mvtec.com* and the cluster of excellence CoTeSys (Cognition for Technical Systems, *www.cotesys.org*)

References

[1] K. Lai, L. Bo, X. Ren, and D. Fox, “Sparse distance learning for object recognition combining rgb and depth information,” in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May, 9–13 2011.

[2] K. Welke, J. Issac, D. Schiebener, T. Asfour, and R. Dillmann, “Autonomous acquisition of visual multi-view object representations for object recognition on a humanoid robot,” in *IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[3] G. Bradski, R. B. Rusu, and K. Konolige, “Opencv: Solutions in perception challenge,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Workshop. Defining and Solving Realistic Perception Problems in Personal Robotics*, October 2010.

[4] M. Ruhnke, B. Steder, G. Grisetti, and W. Burgard, “Unsupervised Learning of 3D Object Models from Partial Views,” in *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, Kobe, Japan, 2009.

[5] L. Montesano, M. Lopes, A. Bernardino, and J. Santos-Victor, “Learning Object Affordances: From Sensory–Motor Coordination to Imitation,” *Robotics, IEEE Transactions on*, vol. 24, no. 1, pp. 15–26, 2008.

[6] B. Drost, M. Ulrich, N. Navab, and S. Ilic, “Model globally, match locally: Efficient and robust 3d object recognition,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010.

[7] A. Hofhauser, C. Steger, and N. Navab, “Edge-based template matching with a harmonic deformation model,” in *Computer Vision and Computer Graphics: Theory and Applications - International Conference VISIGRAPP 2008, Revised Selected Papers*, ser. Communications in Computer and Information Science, A. K. Ranchordas, H. J. Araújo, J. M. Pereira, and J. Braz, Eds., vol. 24. Berlin: Springer-Verlag, 2009, pp. 176–187.

[8] M. Beetz, L. Mösenlechner, and M. Tenorth, “CRAM – A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Taipei, Taiwan, October 18-22 2010, pp. 1012–1017.

[9] D. McDermott, “A Reactive Plan Language,” Yale University,” Research Report YALEU/DCS/RR-864, 1991.

[10] L. Mösenlechner, N. Demmel, and M. Beetz, “Becoming Action-aware through Reasoning about Logged Plan Execution Traces,” in *IEEE/RSJ International Conference on Intelligent RObots and Systems.*, Taipei, Taiwan, October 18-22 2010, pp. 2231–2236.

[11] A. Maldonado, U. Klank, and M. Beetz, “Robotic grasping of unmodeled objects using time-of-flight range data and finger torque information,” in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, October 18-22 2010, pp. 2586–2591.