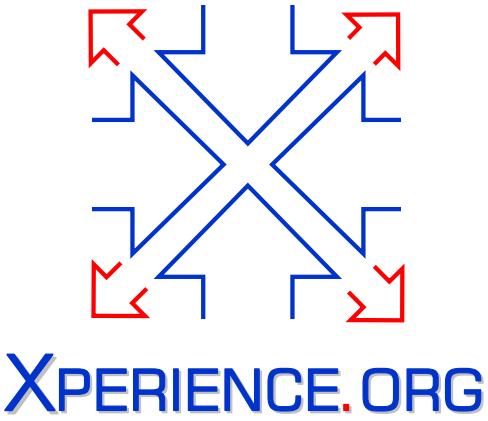
Project Acronym:	Xperience
Project Type:	IP
Project Title:	Robots Bootstrapped through Learning from Experience
Contract Number:	270273
Starting Date:	01-01-2011
Ending Date:	31-12-2015



Deliverable Number:	D5.2.4
Deliverable Title:	Demonstration on Scenario 1: Execution of a recipe.
Type (Internal, Restricted, Public):	PU
Authors:	Alejandro Agostini, Mohamad Javad Aein, Markus Schoeler, Eren Aksoy, Sandor Szedmak, Emre Ugur, Wail Mustafa, Mikkel Tang Thomsen, Dirk Kraft, Norbert Krüger, Justus Piater, Florentin Wörgötter
Contributing Partners:	ALL

Contractual Date of Delivery to the EC:	31-01-2015
Actual Date of Delivery to the EC:	06-05-2015

Contents

Chapter 1

Executive Summary

This deliverable describes the demonstration in scenario 1: preparation of a recipe. The demonstration consists of a robot preparing a salad, where objects in the original recipe might be missing and the robot should be able to figure out which other objects in the scenario can be used to replace the missing ones. In the demonstration we show the integration of perceptual, object, and action categories for the salad making scenario with planning mechanisms to perform plans with a large set of grounded object-action complexes on which generalization processes are performed by structural bootstrapping. The demonstration realizes the main objectives of work package WP5.2.

Fig. 1.1 shows a general diagram with the processes taking place in the demonstration. First, a plan is generated from the original recipe. This original recipe is described as a prototypical planning domain definition that would allow a planner to generate the sequence of instructions (plan) to prepare the salad. Once the plan is generated, the robot checks if all the required objects are in the scenario. If there are missing objects, the robot connects to the Repository of Objects and Attributes with Roles (ROAR) to find out which of the objects perceived in the scene have the same affordances of the missing ones. If replacements are found, the plan is updated and executed using the replacements.

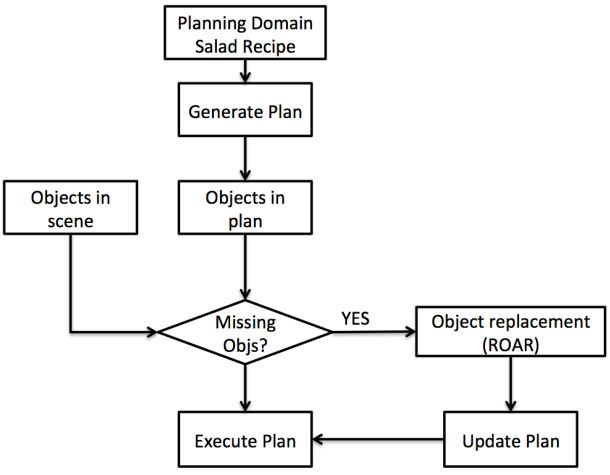


Figure 1.1: General diagram of the processes implemented in the salad making scenario.

The following sections describe the main aspects of the processes involved in the deliverable. Further details can be found in the attached paper [AAS+15] and Deliverable D3.1.3.

1.1 Objects in Scene: Object Recognition Mechanisms

For object recognition in the salad making scenario we use the detection and extraction pipeline proposed by Schoeler et al. [9], which is a fast and automatic system capable to extract and learn unknown objects with minimal human intervention. For this it employs a two-level pipeline combining the advantages of

RGB-D sensors for object detection (see Figure 1.2) and high-resolution cameras for object classification. It projects proposals generated on the low-resolution range image to the higher resolution camera frame. Furthermore, a Radial keypoint orientation scheme is employed, which orients local descriptors (in this case SIFT and CyColor) pointing away from the object's center. Compared to the fixed orientation scheme (often used in scene classification) it is rotation invariant. Compared to the dominant local gradient orientation (e.g. SIFT-Detector, SURF-Detector) it is shape discriminative due to important shape information being encoded in the dominant image gradients on objects. Together with a spatial bag-of-words encoding this leads to highly invariant but discriminative object signatures, which are used for a support-vector-machine with a histogram intersection kernel.

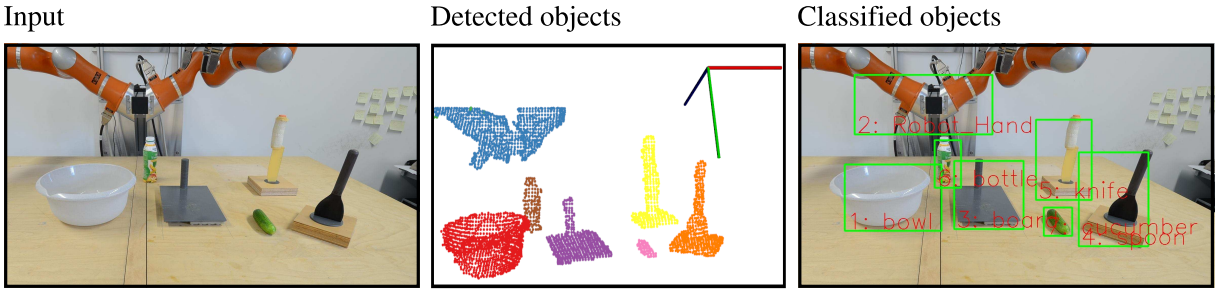


Figure 1.2: Object detection and recognition pipeline trained on objects from the salad making scenario. Left: RGB high-resolution image; Middle: Extracted objects in 3D. The camera view direction is shown by the blue axis; Right: Classification on detected objects.

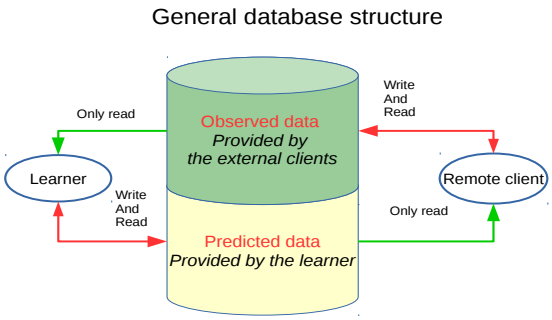
1.2 Repository of Objects and Attributes with Roles (ROAR)

This section presents a brief description of the repository of objects and attributes with roles (ROAR), coding object affordances used for finding replacements. For more details please refer to the attached paper [AAS+15] and Deliverable D3.1.3.

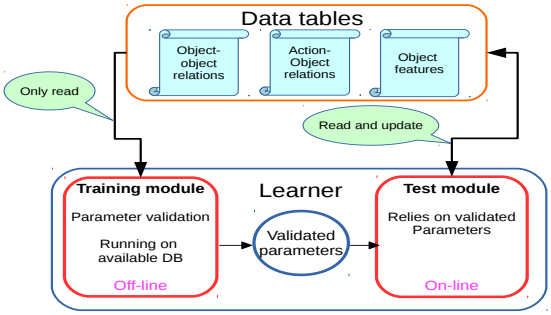
The learning infrastructure of object-action relation and the replacement of the objects or actions with a suitable one is built around the ROAR module. That module behaves as a certain type of object memory where the set of available or potentially available objects together with the affordances related attributes are stored. The ROAR stands for *repository of objects&attributes with roles*. The database of prior knowledge can be created by hand or by prior experience. It allows objects to be retrieved by their attributes, and the attributes of novel objects can be inferred.

The ROAR module serves as an active database system, which not only stores and returns the data items, but via machine learning tools it extends the database with predicted elements, and can provide data not observed earlier by the users connected to the database. This type of active database might also be called as "Intelligent Relational Database". The learning methods working in the background of the ROAR are introduced by [13] and [11]. The first one is built around the Homogeneity Analysis, a Singular Vector Decomposition based method, the second one is a large scale maximum margin based learner. Both are designed to predict missing relations from the available data sources.

ROAR can learn from various data sources and can make reasoning in different ways. While ROAR has the potential of representing any type of relations, it also supports learning from and reasoning on (object, action, score) tuples. Figure 1.3 shows the structure of the ROAR how this intelligent database is connected to different data sources, e.g. from world wide web resources connected as remote clients, and how the reasoning capability of ROAR can be exploited by different modules. First of all, the relations represented in ROAR can be automatically bootstrapped by common sense knowledge extracted from text (action, object, score) tuples. Alternatively, ROAR can learn object-action relations directly from the domain descriptions used by the planners. Note that, ROAR not only stores those relations, but has the ability to infer the scores of the missing ones. In the current scenarios, reasoning capability of ROAR is used when the plan execution fails due to missing target objects, and the plan execution monitoring module searches for alternative objects that can replace the target object. The ROAR can represent categorical and continuous data as well, and have the potential capability to make inferences based on features obtained from perception.



ROAR updates the information accumulated in the database. It works in the background and filling up the unknown data fields, and those ones in which the confidence is low.



The resource intensive training process of the learners can run off-line in the background, and the test module executes the necessary updates on-line.

Figure 1.3: General structure of the ROAR

1.2.1 Intelligent database

The basic concept of combining database techniques with artificial intelligence to improve the performance of a robot has been discussed since the dawn of computers. A reasonable summary of ideas an a comprehensive definition of the intelligent database are published in the book [6]. One of the most recent collection of papers dealing with the possible building blocks can be found in [10]. An integrated general system, *Robobrain*, similar to the ROAR is published by the authors of [8]. Since the ROAR system is built around a broadly used professional open source database system (Postgresql) the required software engineering implementing the connections, the concurrence of the data retrieval and updates could be reduced to minimum. This kind of universal approach can connect working robots world wide and allows us to build an intelligent community via propagation of the locally collected knowledge with only moderate efforts.

1.2.2 Affordance Extraction Based on Vision

Assigning affordances to novel objects has been done using an object categorisation method that utilises a multi-label learning algorithm, namely Joint SVM [14], provided by ROAR. Using this method, objects are described using global 3D features. The features consist of histograms of pairwise geometric measures, namely angle and scale-invariant distance, computed between local features (3D texlets). The way objects described using these features was discussed in more detail in D2.3.3 and [5]. The use of Joint SVM as a multi-label learning algorithm was introduced in the attached technical report [MXK+15].

In this demo, the method was used to assign affordances to missing (novel) objects and store the affordances in ROAR. To demonstrate the performance of this method in the salad making scenario, we tested the method on every object involved in the scenario (e.g. those shown in Sec. 1.1) as well as some new objects. For each object, the object in question is considered missing whereas the method is trained with the rest of the objects. In the training phase, the method is presented with 20 samples for each object. In the prediction phase, the method assigns affordances to the individual object in question (which was not included in the training phase). Note that the objects are labeled with human-defined affordances relevant to this scenario. The labels are composed of two parts describing accordances in ROAR: action and preposition. For each affordance, the two parts are concatenated into a single label such as "[action]_[preposition]"[1].

Table 1.4 shows the outcome of the prediction phase on the objects involved in this demo. The ratios represent the number of test samples (out of 20 samples for each object) to which the specific affordance was assigned. The grey shadows indicate how each object was labeled. For the object 'zucchini' for instance, the method assigned the affordances "cut_null", "drop_null", "pick_null", "pick_place_null", and "place_null" to 19 test samples. These are correct affordances according to how this object is labeled (the grey shadows). The system was able to make these prediction because of the zucchini's visual similarity to 'banana', which was included in this case in the training phase and from which these affordances were learned. Moreover, the method assigned the affordance "pour_from" for one sample, which is an

[1] A "null" preposition means that the action is performed on the object.

incorrect prediction. The object 'board' is labeled with "pick_place_to" and "place_to" affordances but the affordances were never predicted for the 'board' object. This is because the method has never learned such affordances before (i.e,. in the training phase in this case no object had these affordances). To use the extracted affordances presented in Table 1.4 in the demo, we created a Table called *object_action_vision* (see Table 1.2) in the ROAR database.

objects	pick_place_to	place_to	cut_null	drop_null	pick_null	pick_place_null	place_null	pour_from	pour_into	stir_null	drop_into	cut_with	stir_with
banana	0	0	0.55	0.55	0.55	0.55	0.55	0.1	0	0	0	0.45	0.45
board	0	0	0	0	0	0	0	0	0	0	0	1	1
bottle	0	0	1	1	1	1	1	0	0	0	0	0	0
bowl	0	0	0.05	0.05	0.05	0.05	0.05	0	0.95	0.95	0.95	0	0
unknown_bowl_blue	0	0	0	0	0	0	0	0	0.8	0.8	0.8	0	0
cleaver	0	0	0	0	0	0	0	0	0.35	0.35	0.35	0.15	0.5
zucchini	0	0	0.95	0.95	0.95	0.95	0.95	0.05	0	0	0	0	0
jar	0	0	1	1	1	1	1	0	0	0	0	0	0
unknown_knife_orange	0	0	0	0	0	0	0	0	0	0	0	0.8	1
unknown_knife_purple	0	0	0	0	0	0	0	0	0	0	0	1	1
spoon	0	0	0	0	0	0	0	0	0	0	0	1	1
unknown_spoon_orange	0	0	0	0	0	0	0	0	0	0	0	0.9	1

Figure 1.4: The prediction of affordances when the individual object is novel whereas the rest are known.

1.3 Object Replacement and Plan Updating

The object replacement and plan updating mechanisms are triggered in case any of the objects involved in the plan generated from the prototypical planning domain is missing (see Fig. 1.1). In this case, the system extracts from the planning operators (POs) involved in the generated plan all the predicates coding the missing object affordances. Then, the system checks, for each of the missing objects, which object in the current scenario has the same affordances as the missing one. This is done by extracting the affordances of the evaluated object from the ROAR tables (see Tables 1.1 and 1.2) and then checking if these affordances match those of the missing object. If an object in the scenario has all the same affordances, then it is used to replace the missing one. If replacements for all the missing objects are found, then the plan is updated with the replacements. For a more detailed description of this process, please refer to the attached paper [AAS+15].

1.4 Salad Making Scenario: Planning Domain Definition and ROAR tables

In the salad making scenario, the task consists of preparing a cucumber salad by first cutting the cucumber in pieces on a cutting board, dropping these pieces into a bowl, pouring salad dressing into the bowl, and then stirring everything with a spoon.

The prototypical planning domain definition for this scenario involves the following objects:

- Robot hand,
- table,
- board (cutting board),
- cucumber,
- knife,
- bottle (containing the salad dressing),
- bowl,
- spoon.

In the planning domain definition we consider the name for the predicates associated to affordances in the ROAR tables, e.g. $cutObj(cucumber)$ indicating that the cucumber is an object that can be cut. In addition to the predicates coding affordances, the domain definition also comprises predicates describing the relations between objects, e.g. cucumber on the board: $on(cucumber, board)$, and the object status, e.g. cucumber not cut: $!cut(cucumber)$.

Some of the predicates considered in the initial state of the planning problem definition for the salad making scenario are: $on(cucumber, table)$, $!on(cucumber, board)$, $free(hand)$, $PPto(board)$, $!cut(cucumber)$, $cutWith(knife)$, $!in(cucumber, bowl)$, $!stirred(bowl)$, $stirWith(spoon)$, and $pourObj(dressing)$. The predicates for the goal specification for the task of preparing a cucumber salad are: $cut(cucumber))$, $in(dressing, bowl))$, $in(cucumber, bowl))$, and $stirred(bowl))$, indicating that the cucumber should be cut, the dressing and the cucumber should be in the bowl, and the bowl should be stirred, respectively.

For plan generation we use the logic-based planner PKS [7]. The PKS description of the planning operators defined for the salad making scenario are presented in Fig. 1.5.

```
action pick_place( ?obj, ?with, ?from, ?to){
   preconds:
         K(PPobj(?obj)) &
         K(on(?obj, ?from)) &
         K(!on(?obj, ?to)) &
         K(PPwith(?with)) &
         K(free(?with)) &
         K(PPto(?to))
   effects:
         add(Kf, !on(?obj,?from)),
         add(Kf, on(?obj,?to))   }
```

```
action drop( ?obj, ?with, ?from, ?in){
   preconds:
         K(dropObj(?obj)) &
         K(on(?obj, ?from)) &
         K(!in(?obj, ?in)) &
         K(dropWith(?with)) &
         K(free(?with)) &
         K(dropInto(?in))
   effects:
         add(Kf, !on(?obj, ?from)),
         add(Kf, in(?obj, ?in)) }
```

```
action pour( ?obj, ?with, ?from, ?in){
   preconds:
         K(pourObj(?obj)) &
         K(in(?obj, ?from)) &
         K(!in(?obj, ?in)) &
         K(purWith(?with)) &
         K(free(?with)) &
         K(pourInto(?in))
   effects:
         add(Kf, !in(?obj, ?from)),
         add(Kf, in(?obj, ?in)) }
```

```
action cut( ?obj, ?with, ?on){
   preconds:
         K(cutObj(?obj)) &
         K(on(?obj, ?on)) &
         K(!cut(?obj)) &
         K(cutWith(?with)) &
         K(cutOn(?on))
   effects:
         add(Kf, cut(?obj))   }
```

```
action stir( ?obj, ?with){
   preconds:
         K(stirObj(?obj)) &
         K(!stirred(?obj)) &
         K(stirWith(?with))
   effects:
         add(Kf, stirred(?obj)) }
```

Figure 1.5: Planning operators in PKS notation defined for the salad making scenario. The predicates coding affordances compatible with the ROAR are marked in red.

We created a SQL database for the salad making scenario using the interface software *Postgresql*. The name of the database is *salad–scenario*, and the main table used to find object replacement is called *object–action*, which was created manually. Table 1.1 presents some elements of this table. On the other hand, Table *object-action-vision* (see Table 1.2) presents some of the affordances extracted from visual features, as explained in Sec. 1.2.2. This table is used in case the system is not able to find a replacement using the manually coded table *object-action*.

The fields of the tables are the object name, the action in which it is involved, the preposition, indicating the specific function of the object in that action, and the score, indicating how probable is that the object can be used for the corresponding action. In addition, the table contains a field with the name of the predicate that will be used to code the corresponding affordance in the planning domain definition.

1.5 Plan Execution

In the execution phase of the planning framework, we created a library of actions that encodes the abstract semantic structure of manipulations. The derived structure allows robots to execute various chains of human-like manipulation actions, such as the ones involved in *preparing a salad*.

In the action library, actions are represented in the semantic level by employing the concept of Semantic Event Chains (SECs) introduced in [2]. SECs capture the essence of an action by employing computer vision techniques. Given a human demonstrated action, SECs segment and track all objects exist in the scene and encodes the entire action as a matrix in the spatiotemporal domain. Each column in the SEC matrix represents a decisive temporal anchor point in the manipulation, i.e. indicates a unique and descriptive scene "state". All rows in the SEC are the abstract spatial relations between object pairs. Fig. 1.6 pictures the extracted SEC matrix from a robot execution of a *pick_place* manipulation. Since each state in the SEC corresponds to a topological change in the manipulation, we consider each transition from one SEC column to the next as a movement primitive, such as *approach* or *grasp*. In Fig.1.6, the necessary primitives associated with each column of the SEC matrix are shown for the *pick_place* example.

Table 1.1: Object-Action Table. Salad-Scenario Database.

Object	Action	Preposition	Predicate	Score
cucumber	cut	null	cutObj	1
carrot	cut	null	cutObj	1
banana	cut	null	cutObj	1
knife	cut	with	cutWith	1
cleaver	cut	with	cutWith	1
cucumber	drop	null	dropObj	1
carrot	drop	null	dropObj	1
banana	drop	null	dropObj	1
board	drop	from	dropFrom	1
cucumber	pick place	null	PPObj	1
carrot	pick place	null	PPObj	1
banana	pick place	null	PPObj	1
table	pick place	from	PPFrom	1
board	pick place	to	PPto	1
spoon	stir	with	stirWith	1
knife	stir	with	stirWith	1
bowl	stir	null	stirObj	1
bowl	pour	into	pourInto	1
bowl	drop	into	dropInto	1

Table 1.2: Object-Action-Vision Table. Salad-Scenario Database.

Object	Action	Preposition	Predicate	Score
zucchini	cut	null	cutObj	0.95
banana	cut	null	cutObj	0.55
unknown_knife_orange	cut	with	cutWith	0.8
unknown_knife_purple	cut	with	cutWith	1
zucchini	drop	null	dropObj	0.95
banana	drop	null	dropObj	0.55
zucchini	pick place	null	PPObj	0.95
banana	pick place	null	PPObj	0.55
spoon	stir	with	stirWith	1
spoon	cut	with	stirWith	1
unknown_spoon_orange	stir	with	stirWith	1
unknown_spoon_orange	cut	with	stirWith	0.9
bowl	stir	null	stirObj	0.95
bowl	pour	into	pourInto	0.95
bowl	drop	into	dropInto	0.95
unknown_bowl_blue	stir	null	stirObj	0.8
unknown_bowl_blue	pour	into	pourInto	0.8
unknown_bowl_blue	drop	into	dropInto	0.8

Note that these primitives are symbolic, but, on the other hand, are fully grounded at the signal level with uniquely tracked image segments. In [3] we also show that those symbolic action primitives are learned from human demonstrations in an unsupervised manner.

During the learning from human demonstrations, we also enrich the raw symbolic SEC primitives with additional object and trajectory information. Each image segment is classified as *manipulator*, *primary* and *secondary objects* by considering their exhibited roles in the action as described in [3]. *Manipulator* is the main actor that performs the planned goal in the action, e.g. hand. *Primary object*, e.g. knife, is the one that is directly manipulated by the manipulator. All other objects interacting with the primary object, are called *secondary objects*, e.g. cucumber to be cut. We next identify the classified image segments by employing the object recognition method in [9]. We additionally capture the trajectory pattern of the detected manipulator, e.g. human hand, with the modified Dynamic Movement Primitives (DMPs, [4]) and attach it to the respective primitive in the SEC. Fig. 1.6 shows detected and recognized

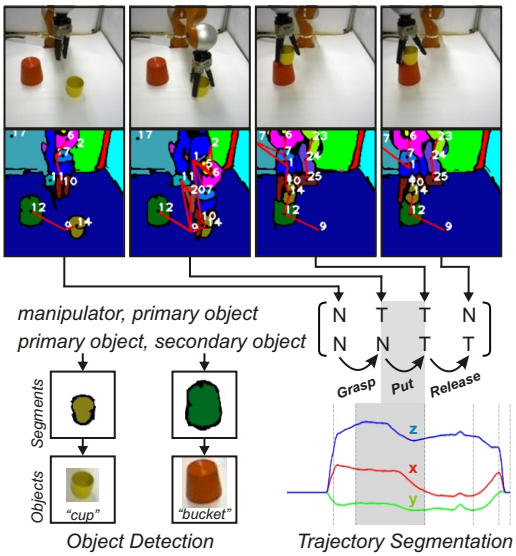


Figure 1.6: Robot execution of a *pick_place* manipulation. The snapshots of the performed action together with segmented images are shown on the top. The symbolic graph sequence is given in the middle. Each graph corresponds to one column in the SEC matrix given in the middle together with the corresponding action primitives. The valid symbolic entries in the SEC matrix are the spatial object relations, i.e. N (Not touching) and T (Touching). In the bottom, recognized objects in the SEC and the attached segmented trajectory profile for the robot manipulator are given.

primary and *secondary objects* in the current scene. The trajectory profile depicted in Fig. 1.6 is the estimated *manipulator* movement from a sample human demonstration. We now segment the whole trajectory at the anchor points in the SEC and feed back to the robot for the sequential execution of each primitive, e.g. the *Put* primitive is shown in gray box.

Once the SEC representation is augmented with action descriptive object and trajectory parameters, we employ the Finite State Machine (FSM) introduced in [1]. The FSM creates one state for each column of SEC matrix and allows the robot to transit from one primitive the next by applying the embedded trajectory pattern to the primary object in the plan. The input of FSM is the relation of objects and its output are the primitives. To detect the spatial relations in the current scene, the robot uses the combination of proprioceptive (e.g. position) and exteroceptive (e.g. tactile, force, and vision) sensors and sends an error signal if the desired primitive, i.e. expected effect in the spatial relations, is not observed.

1.5.1 Task Specific Grasping of Unknown Objects

For transfer of actions between objects we investigated the space of visual triggered action affordances in terms of a combined feature-action space (as described in D2.3.3 and [12]). We spatially combine visual features with actions, both from a simulated environment in terms of simulated visual sensors and simulated grasps. We vary important dimensions in the visual space such as the granularity (size) of the surface patches, the order (one or two combined features) and the feature abstraction (a surface patch, a surface patch with a border label and a surface patch with border label and direction to the border). In our search for promising regions (visual triggered affordances) in the space, we perform a neighbourhood analysis and extract the success-probability and the amount of support of the different feature-action particles and save the results in a database. This database enables grasp predictions on previously unseen objects.

Based on the method, we investigated the visual action space as described above and show how the ability to make grasp predictions is strongly depending on the visual descriptor used. In particular it is seen how the addition of a border label and direction improve the performance significantly for specific object categories such as bowls, cups and knives, essentially extracting parts of the structure properties that these objects possess.

In D2.3.3 and [12] we focused our work on general grasping of unknown objects. For the integration into

the Salad making scenario we extended this to task specific grasping (e.g., grasping a knife to later on use it for cutting). This years demonstration is an extension of the standalone grasping demo shown in year three (D5.2.3). We now integrated the grasping of unknown knives work into the general scenario on the UGOE platform.

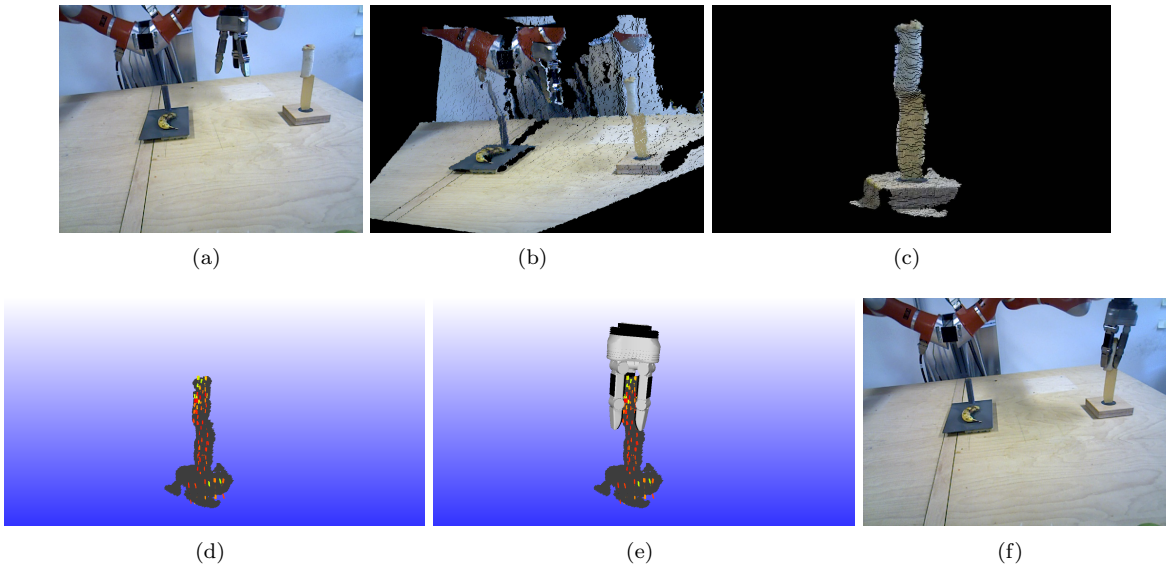


Figure 1.7: Exemplified grasp process. (a) Starting scene with the unknown knife to the right, (b) RGB-D representation of the scene, (c) segmented knife, (d) predicted grasp positions, (e) best predicted grasp, (f) grasp execution.

We show a standalone video of the grasping for cutting in **GraspingForCutting.mp4**, see also Figure 1.7. There we present the visual representation, the predicted grasps and the execution of the best grasp. In addition, we show our two stage grasp generation process with (a) generation of grasps for known knives and (b) application of these grasped knives for a cutting task. Based on simulated successes and failures we create a model relating visual features to action success that is used for the predictions later.

1.6 Demonstrated Cases

The demonstration in scenario 1 comprises three cases (see attached video **D5_2_4.m4v**). In the first case, all the objects involved in the plan generated from the prototypical planning problem definition are present in the scenario. We use this case as the reference case. The second case shows how the system is able to find a replacement for the main object involved in the preparation of the salad, i.e. the cucumber. This is evaluated by extracting the cucumber and placing a banana and other objects on the scene. In this case, it is expected that the system is able to identify the banana as a valid substitution of the cucumber for the preparation of a salad. The last case demonstrates how the extraction of affordances based on visual features can be used to find object replacements. To this end, we extract the grey bowl from the scene, used in several actions of the plan, and place a new bowl having different shape and color, which is *unknown* by the system [2].

For all the cases, the original plan, generated from the prototypical planning problem definition is:

$$\begin{aligned} &pick\_place(cucumber, hand, table, board)\\ &cut(cucumber, knife, board)\\ &drop(cucumber, hand, board, bowl)\\ &pour(dressing, hand, bottle, bowl)\\ &stir(bowl, spoon)). \end{aligned} \tag{1.1}$$

[2] An object is assumed to be unknown when it is not considered in the prototypical planning domain definition nor in the ROAR table generated manually (Table 1.1)

1.6.1 Case 1: No Missing Objects

In the first case, where no missing objects are in the scene, the original plan (1.1) can be successfully executed without the need of finding object replacements. Figures 1.8 presents a snapshot of the scenario for this case. Further details can be found in the attached video.

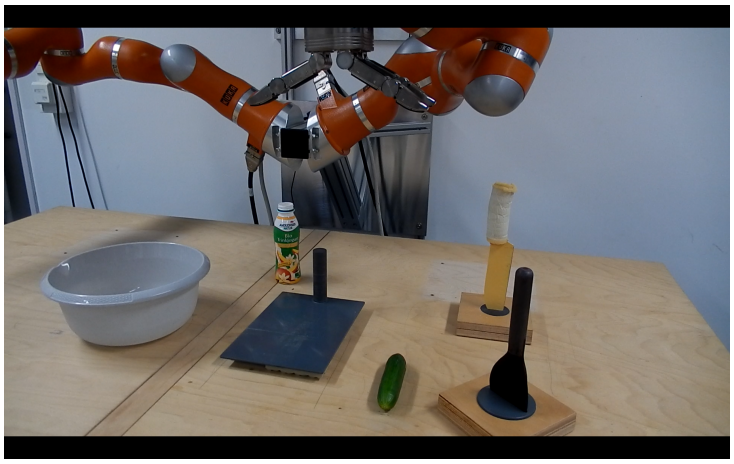


Figure 1.8: Scenario for the first case of the demo: no missing objects.

1.6.2 Case 2: Cucumber Missing, Replace it with Banana.

The second case of the demo requires finding a replacement for the cucumber, which was not included in the scenario. The objects in the scene for the second case are (see Fig. 1.9):

- Robot hand,
- table,
- board,
- banana,
- jar,
- knife,
- bottle,
- bowl,
- spoon.

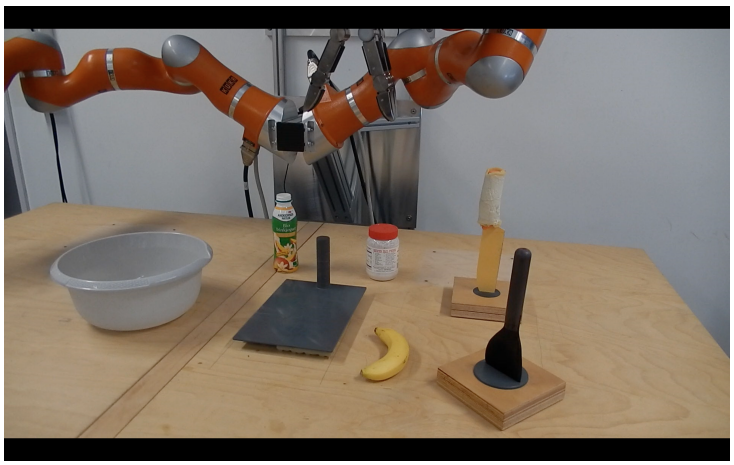


Figure 1.9: Scenario for the second case of the demo: cucumber missing.

The system identifies the missing cucumber and extracts the affordances associated to it from the POs involved in the original plan 1.1 (see also Fig. 1.5): $PPobj(cucumber)$, $cutObj(cucumber)$, and

$dropObj(cucumber)$, indicating that the cucumber is an object that can be picked and placed, cut, and dropped, respectively.

Then, the system evaluates which object in the scene has all the three affordances of the missing cucumber using Table 1.1. In this case, the banana presents the same affordances and is used to replace the cucumber in the plan:

$$\begin{aligned} &pick\_place(banana, hand, table, board)\\ &cut(banana, knife, board)\\ &drop(banana, hand, board, bowl)\\ &pour(dressing, hand, bottle, bowl)\\ &stir(bowl, spoon)). \end{aligned} \tag{1.2}$$

The updated plan 1.2 was successfully executed by the robot as shown in the video.

1.6.3 Case 3: Bowl Missing, Replace it with Unknown Object.

The last case of the demo shows how the extraction of affordances based on visual features (Sec. 1.2.2) is used to find a replacement of a missing object. In this case, we extract from the scene the grey round bowl, which is identified as *bowl* and known by the system, and place a blue square bowl that is not included in the planning problem definition, nor in the hand-coded ROAR Table 1.1, i.e. which is not known by the system (see Fig. 1.10).

In this case, the system identifies the missing bowl and extracts the predicates coding affordances from the POs involved in the generated plan (see plan 1.1 and Fig. 1.5): $dropInto(bowl)$, $pourInto(bowl)$, and $stirObj(bowl)$, indicating that the bowl can be used to drop things into, pour things into, and can be stirred. Then, the system searches for replacements in the hand-coded ROAR Table 1.1. Since no replacements are found in this table, the system then looks in the table coding affordances extracted from visual features (Table 1.2). This table actually contains an object having the same three affordances of the missing bowl: the *unknown_bowl_blue* object.

Finally, the system uses the found object *unknown_bowl_blue* to replace the bowl in plan 1.1, leading to plan:

$$\begin{aligned} &pick\_place(cucumber, hand, table, board)\\ &cut(cucumber, knife, board)\\ &drop(cucumber, hand, board, unknown\_bowl\_blue)\\ &pour(dressing, hand, bottle, unknown\_bowl\_blue)\\ &stir(unknown\_bowl\_blue, spoon)), \end{aligned} \tag{1.3}$$

The updated plan 1.3 was successfully executed as shown in the video.

Figure 1.10: Scenario for the third case of the demo: grey bowl missing.

Chapter 2

Conclusions

In this deliverable we show the integration of several components for the demonstration of Scenario 1: preparation of a recipe. This integration comprises mechanisms for object recognition (Sec. 1.1), planning (Sec. 1.4), affordances extraction and object replacement (Sections 1.2 and 1.3), and robot manipulation, for the actual execution of actions (Sec. 1.5).

For the demonstration we used the case of study of a preparation of a cucumber salad. The results showed that the enumerated mechanisms were successfully integrated for the updating and execution of plans using structural bootstrapping for missing object replacements.

References

[1] M. J. Aein, E. E. Aksoy, M. Tamosiunaite, J. Papon, A. Ude, and F. Wörgötter. Toward a library of manipulation actions based on semantic object-action relations. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 4555–4562, 2013.

[2] 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.

[3] E. E. Aksoy, M. Tamosiunaite, and F. Wörgötter. Model-free incremental learning of the semantics of manipulation actions. *Robotics and Autonomous Systems (RAS) (In press)*, 2015.

[4] T. Kulvicius, K. J. Ning, M. Tamosiunaite, and F. Wörgötter. Joining movement sequences: Modified dynamic movement primitives for robotics applications exemplified on handwriting. *IEEE Transactions on Robotics*, 28(1):145–157, 2012.

[5] Wail Mustafa, Dirk Kraft, and Norbert Krüger. Extracting categories by hierarchical clustering using global relational features. In *Iberian Conference on Pattern Recognition and Image Analysis (IbPRIA), the 7th*, 2015. (accepted).

[6] Kamran Parsaye, Mark Chignell, Setrag Khoshafian, and Harry Wong. *Intelligent Databases: Object-oriented, Deductive Hypermedia Technologies*. John Wiley & Sons, Inc., New York, NY, USA, 1989.

[7] R. Petrick and F. Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In Malik Ghallab, Joachim Hertzberg, and Paolo Traverso, editors, *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS-2002)*, pages 212–221. AAAI Press, 2002.

[8] Ashutosh Saxena, Ashesh Jain, Ozan Sener, Aditya Jami, Dipendra K. Misra, and Hema S. Koppula. Robobrain: Large-scale knowledge engine for robots. *arXiv*, Artificial Intelligence, Robotics, 1412.0691v1, 2014.

[9] M. Schoeler, S. Stein, J. Papon, A. Abramov, and F. Wörgötter. Fast self-supervised on-line training for object recognition specifically for robotic applications. In *International Conference on Computer Vision Theory and Applications (VISAPP)*, January 2014.

[10] Myra Spiliopoulou, Lars Schmidt-Thieme, and Ruth Janning, editors. *Data Analysis, Machine Learning and Knowledge Discovery*. Springer, 2014. http://link.springer.com/book/10.1007%2F978-3-319-01595-8.

[11] S. Szedmak, E. Ugor, and J.Piater. Knowledge propagation and relation learning for predicting action effects. In *Proceedings of the IEEE Intl. Conf. on Intelligent Robots and Systems (IROS 2014), Chicago*. 2014.

[12] Mikkel Tang Thomsen, Dirk Kraft, and Norbert Krüger. Identifying relevant feature-action associations for grasping unmodelled objects. *Paladyn. Journal of Behavorial Robotics*, 6(1):85–110, 2015.

[13] Hanchen Xiong, Sandor Szedmak, and Justus Piater. Homogeneity analysis for object-action relation reasoning in kitchen scenarios. In *2nd Workshop on Machine Learning for Interactive Systems, (Workshop at IJCAI)*, page 3744. 2013.

[14] Hanchen Xiong, Sandor Szedmak, and Justus Piater. Scalable, accurate image annotation with joint svm and output kernels. *Neurocomputing*, 2015. (accepted).

Attached Articles

[AAS+15] A. Agostini, M. J. Aein, S. Szedmak, E. Aksoy, J. Piater, and F. Wrgtter. Using Structural Bootstrapping for Object Substitution in Robotic Executions of Human-like Manipulation Tasks. In *IROS*, 2015. Submitted.

[MXK+15] Wail Mustafa, Hanchen Xiong, Dirk Kraft, Sandor Szedmak, Justus Piater, and Norbert Krüger. Multi-label object categorization using histograms of global relations. Technical Report 2015–2, Cognitive and Applied Robotics Group, The Maersk Mc-Kinney Moller Institute, University of Southern Denmark, 2015.