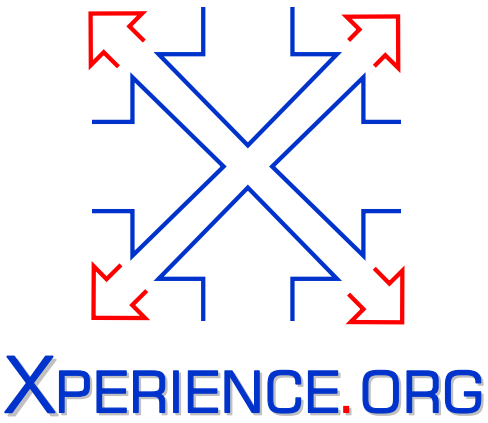
Project Acronym:	Xperience
Project Type:	IP
Project Title:	Robots Bootstrapped through Learning from Experience
Contract Number:	270273
Starting Date:	01-01-2011
Ending Date:	31-12-2015



Deliverable Number:	D5.1.1
Deliverable Title:	Complete Open Source of the developed functionalities
Type (Internal, Restricted, Public):	PU
Authors:	Giorgio Metta, Tamim Asfour, Ekaterina Ovchinnikova, Mikro Wächter, David Schiebener, Simon Ottenhaus, Bojan Nemec, Aleš Ude, Norbert Krüger, Dirk Kraft, Chris Geib, Ron Patrick, Florentin Wörgötter, Alejandro Agostino
Contributing Partners:	ALL

Contractual Date of Delivery to the EC:	31-12-2015
Actual Date of Delivery to the EC:	12-02-2016

Contents

Executive Summary

This deliverable reports on the activities carried out by the Consortium to integrate the skills realized by providing the foundations of a model of cognition that incorporates development, sensori-motor coordination, affordances and high-level prediction, interaction and communication in terms of structural bootstrapping.

The main goals of the Xperience repository is to lay the foundation for interfacing existing components with the newly developed code, by providing infra-structure for code repository and team collaboration. Specific guidelines regarding the production of code and methods to ensure smooth integration among partners have been promoted. This was done in order to achieve significant high-quality standards and the construction of quality of open source software, in particular if they will be used to transfer our findings to the community and used in practical applications.

The document outlines the organization of the Xperience software repository, and gives an overview and summary/documentation of the developed functionalities.

Chapter 1

Complete Open Source of the developed functionalities

1.1 Structure of the Xperience repository

The Xperience software repository is openly accessible at the sourceforge portal at the link `https://sourceforge.net/projects/xperience/` and maintained under SVN versioning control system. Importantly, as clearly declared in the gpl.txt file stored in the repository, the whole software adheres to GNU General Public License (GPL), so that it can be exchanged and used by each Consortium partner and any external third party without any limitation.

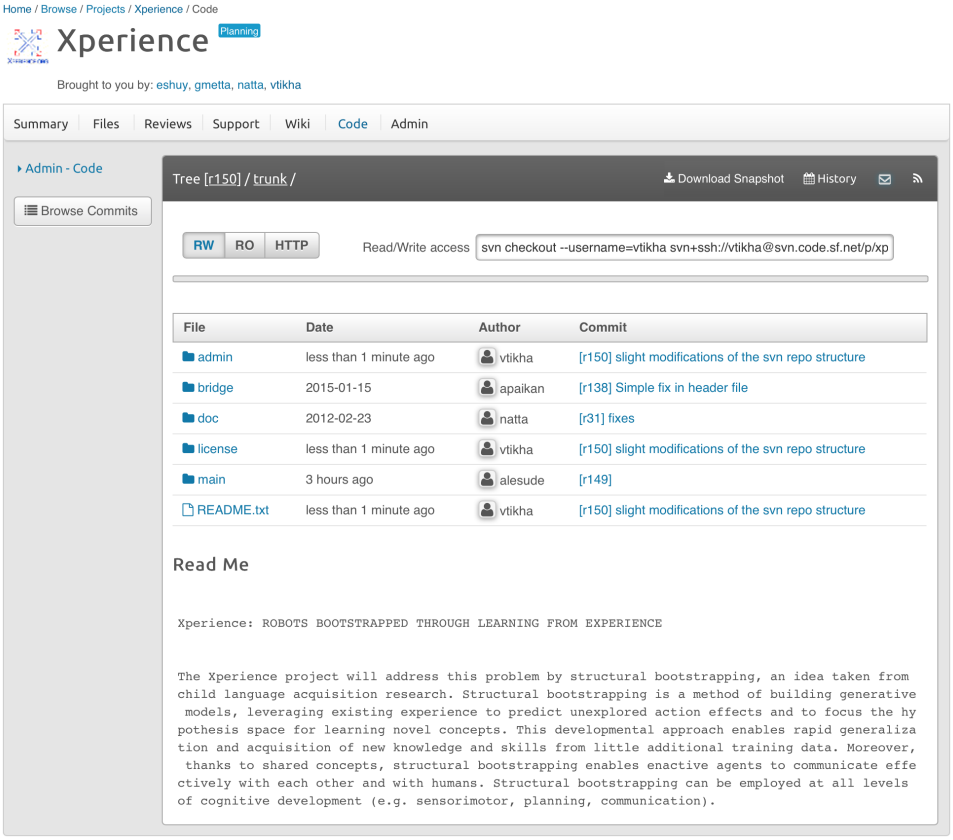


Figure 1.1: Snapshot of the Xperience software repository as it can be browsed online at Sourceforge (snapshot taken on 09/01/2016).

The repository provides a main branch where directories specifically devoted to contain libraries and modules are located 1.1. The main branch contains five directories where:

- **admin**: Stores the general repository scripts for producing the documentation and licensing are kept
- **bridge**: Stores all the code related to the bridge that allows a seamless transition of modules between the two robotic platforms of the Xperience project.
- **doc**: Stores the available software documentation
- **license**: Stores the repository license
- **main**: Stores all the Xperience code that can be profitably shared and reused including relevant libraries and Matlab files.

The Xperience repository represents therefore the place where we share the contributions of the research efforts carried out by all partners.

1.2 Developed functionalities

1.2.1 Release of ArmarX as software framework

ArmarX is a robot development environment which has been developed in Xperience to support the interaction between high-level planning, memory and sensorimotor execution. ArmarX was released officially in this year as an open source project to the public under GPLv2.0 license and is available in the Xperience repository as well as in the development version on gitlab.com/ArmarX. Extensive documentation about ArmarX can be found at http://armarx.humanoids.kit.edu/ as well as self compiled from the source. All ArmarX packages are available in the Xperience svn at the path:

```
main/src/armarx
```

The available ArmarX packages are:

- ArmarXCore - middleware and statecharts
- ArmarXDB - database snapshot
- ArmarXDoc - umbrella documentation package
- ArmarXGui - gui libraries and general gui plugins
- ArmarXSimulation - physics simulation
- MemoryX - robot memory
- RobotAPI - basic robot interfaces
- RobotComponents - advanced robot components
- RobotSkillTemplates - basic robot skills
- SpeechX - speech recognition and language understanding
- Spoac - symbolic planning package
- Tutorials - Tutorial implementations for reference
- VisionX - RGB and RGBD image processing
- Armar3 - application configurations and robot specific skills for Armar3

1.2.1.1 Installing ArmarX

ArmarX is supported for Ubuntu 14.04 and the following packages need to be installed:

```
sudo apt-get install git gitk git-gui mcpp \
libeigen3-dev libcoin80-dev python-psutil \
libsoqt4-dev libdb5.1-dev zeroc-ice35 \
cmake cmake-gui g++ libboost-all-dev \
libdc1394-22-dev doxygen libgraphviz-dev libqwt-dev \
libpcre3-dev curl libcv-dev libhighgui-dev libcvaux-dev \
mongodb libjsoncpp-dev libssl-dev libv4l-dev python-argcomplete \
libopencv-gpu-dev libopencv-photo-dev libopencv-ts-dev \
libopencv-superres-dev libopencv-stitching-dev libopencv-videostab-dev \
libgstreamer-plugins-base0.10-dev cppcheck lcov \
 astyle libopencv-dev freeglut3-dev
```

Additionally, you need to install simox: `https://gitlab.com/Simox/simox/wikis/Installation`

Register the ArmarX base dir in an environment variable:

```
export ArmarX_DIR={your_path_to_xperience_svn}/main/src/armarx
```

Before starting the compilation process, ArmarXCore needs to be cmaked to generate the CLI tool of ArmarX:

```
cd ${ArmarX_DIR}/ArmarXCore/build && cmake ..
```

Now, the armarx-dev CLI tool can build an ArmarX package and all its dependencies, in this case the Armar3 package:

```
${ArmarX_DIR}/ArmarXCore/build/bin/armarx-dev build Armar3
```

This will take some time and about 2GB of RAM per processor core. If you do not have enough RAM you can specify the number of jobs with $-jX$ passed to the build command.

After successful compilation, run cmake in the Armar3 package and its dependencies to generate the scripts for executing scenarios:

```
${ArmarX_DIR}/ArmarXCore/build/bin/armarx-dev build Armar3 --cmake
```

Next, import the MemoryX database snapshot for Armar3:

```
${ArmarX_DIR}/MemoryX/build/bin/mongod.sh start

${ArmarX_DIR}/MemoryX/build/bin/mongoimport.sh \
${ArmarX_DIR}/ArmarXDB/data/ArmarXDB/dbexport/memdb
```

style=bash

As a last step, the Ice middleware on which ArmarX relies needs to be started:

```
${ArmarX_DIR}/ArmarXCore/build/bin/armarx start
```

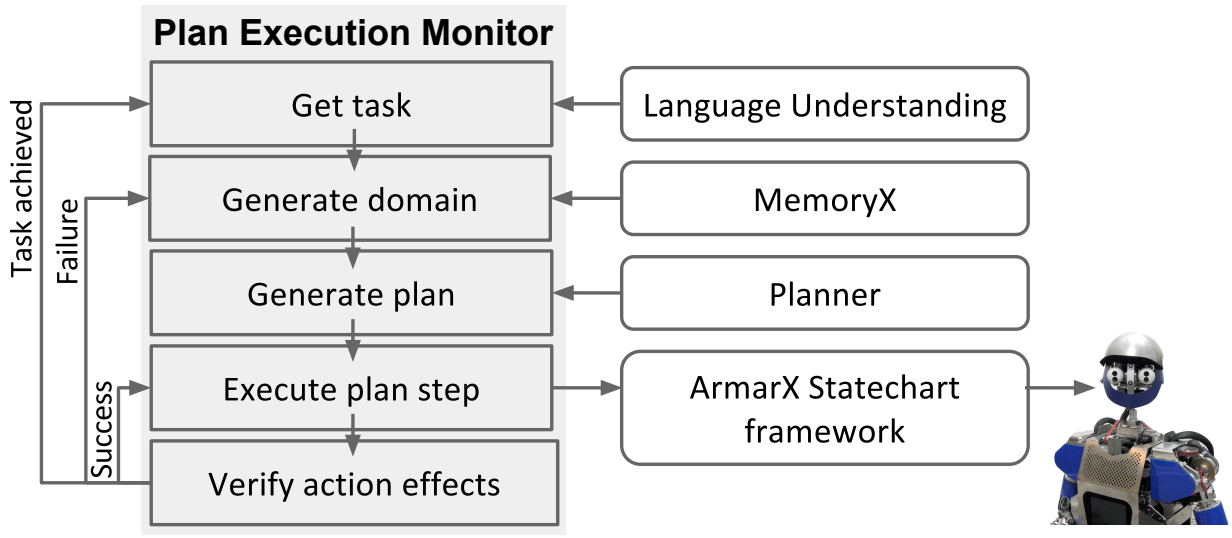


Figure 1.2: Plan generation, execution, and monitoring.

Now, ArmarX application or scenarios can be started. The ArmarX physics simulation is a good starting point and can be started with:

```
cd ${ArmarX_DIR}/ArmarXSimulation/scenarios/Armar3Simulation
./startScenario.sh -w
```

1.2.1.2 ArmarX with PKS and ELEXIR

The planning libriaries PKS and ELEXIR from UEDIN have been integrated into the custom ArmarX package SPOAC (Symbolic Planing with Object Action Complexes). Both libraries have been equipped with an Ice interface and can provide their services as an ArmarX component. Both components can be used directly or implicitly within SPOAC architecture for executing complex tasks with a robot. The components can be found at:

```
${ArmarX_DIR}/Spoac/source/Spoac/components/PandR/PlannerController
```

and

```
${ArmarX_DIR}/Spoac/source/Spoac/components/PandR/RecognizerController
```

The components have also been integrated into a plan execution monitor (PEM, see Figure 1.2) which manages and monitors the execution of a task. The PEM receives a new task from an external trigger, e.g. a speech command, generates a domain from the current world state and consults the planner PKS for a solution to the task problem. If a plan was found, the plan is executed by ArmarX statecharts on a real robot or in a simulation.

The demonstration scenario 2 can be run with (if the simulation is running beforehand as described above):

```
cd ${ArmarX_DIR}/Armar3/scenarios/PlanServer && ./startScenario.sh -w
```

1.2.1.3 ArmarX with Language Understanding

The language understanding pipeline Xperience_LU_pipeline from KIT has been integrated into the custom ArmarX package SpeechX. The pipeline provides its services as an ArmarX component called LanguageUnderstanding (LU) found at

```
${ArmarX_DIR}/SoeechX/source/SpeechX/components/LanguageUnderstanding
```

The component has been integrated into a plan execution monitor (PEM, see Figure 1.2), which manages and monitors the execution of a task. LU receives a text string returned by a speech recognizer, processes it, and passes the result to PEM that directly executes the command, updates the working memory MemoryX, or triggers a planner/plan recognizer. Now, the robot is listening for new speech commands, which can be given via a microphone or the DialogInputWidget in the ArmarX Gui.

1.2.2 Robotic Middlewares: YARP and ArmarX

YARP (see 1.2.10) and the ArmarX (see 1.2.1) are two robotic middlewares which are independently developed based on componentbased and distributed software architecture techniques. Both middle wares support data streaming based on the publishsubscribe paradigm and Remote Procedure Calls (RPC) for intercomponent communication but still differ in the way they implement these functionalities. While ArmarX relies on the ZeroC Internet Communication Engine (ICE), YARP aims at abstracting communication from the underlying protocol (known as YARP carriers) which implements data transfer among the end points. However, the available carriers in YARP do not support communication in an ICEbased network. YARP and ArmarX also differ in the RPC (services) implementations. YARP uses the Apache Thrift IDL format and syntax with its own native serialization, while ArmarX employs the Slice IDL to define interfaces and classes in a network transparent manner. For these reasons the software components from one middleware cannot communicate with those implemented in the other framework and the two middlewares need to be bridged. The software bridging interfaces can be freely accessed via http://svn.code.sf.net/p/xperience/code.

1.2.2.1 Bridging YARP to ArmarX

1.1 summarizes the interfaces (units) which have been developed to bridge YARP (see 1.2.10) to ArmarX framework. The bridging interfaces are implemented as YARP plug ins (port monitor) or separate components which allows software components from ArmarX framework to transparently control the iCub robot. Following we describe the functionality of each interfaces.

Component/Plug-in	Description
YarpImageProvider	Converting YARP image into VisionX image
YarpKinematicUnit	Accessing YARP motor control from ArmarX
YarpTCPController	Accessing iCub Gaze/Cartesian Controller from ArmarX
YarpHandUnit	Accessing iCub hand from ArmarX
YarpPlatformUnit	Accessing iCub mobile platform from ArmarX

Table 1.1: Yarp to ArmarX bridging units.

- **YarpImageProvider**. Vision based software components in ArmarX framework requires specific image format (i.e., using visionX system) which differs from the Yarp image (e.g., PixelRgb). To be able to access the iCub camera images from the ArmarX framework, the YarpImageProvider interface has been developed which implements a VisionX image capturing system by taking the YARP image frames and converting them into the VisionX image system. This interface can be configured to provide the images in mono or stereo format (i.e., iCub left and right cameras) along with providing the iCub stereo camera calibration parameters for the ArmarX system.

- **YarpKinematicUnit**. Implements an ArmarX Kinematic unit/interface to control the iCub joints (i.e., YARP motor control) from the ArmarX system. It combines multiple YARP motor interfaces such as IPositionControl, IVelocityControl and ITorqueControl into a single ArmarX kinematic interface to control the iCub joints respectively in position, velocity and torque mode. Moreover, using this unit, the iCub motor encoders (IEncoders) can be accessed from the ArmarX framework. 1.3. demonstrates an example of controlling iCub (in the simulator) using the ArmarX motor GUI.

- **YarpTCPControl**. Direct and inverse kinematic of a robot in ArmarX framework is directly calculated from a given model. However, to facilitate controlling the iCub robot end effectors (i.e., left and right arm) in Cartesian mode from the ArmarX framework, the YarpTCPControl units has been implemented which uses the iCub Cartesian solver and controller. Moreover, this unit allows accessing verity of the iCub Gaze controller functionalities such as controlling the iCub eyes to gaze at a specific 3D point in space.

- **YarpHandUnit**. All the iCub joints can be controlled using the YarpKinematicUnit. That also includes independent joints of the iCub fingers. However, the current implementation of the hand controller unit in ArmarX framework consider only one degree of freedom for each hand (opening and closing). This bridging units provides a simple opening/closing interface to the iCub hand within the ArmarX framework using predefined joint positions.

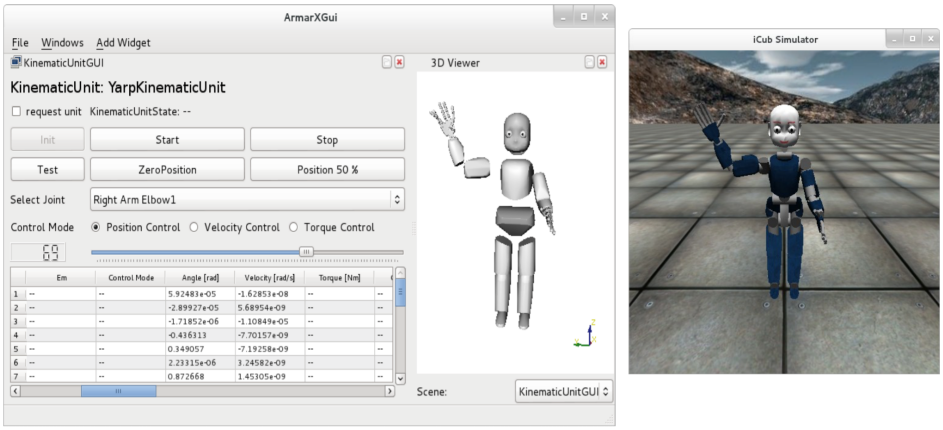


Figure 1.3: An example of controlling iCub (in the simulator) from the ArmarX motor GUI.

- **YarpPlatformUnit**. Some of the software components from the ArmarX framework require the position of the robot mobile platform in a known coordination. Although this is not required for the most of the applications with iCub (i.e., they consider iCub as a fixed platform), the YarpPlatformUnit has been developed to provide a fix position of the iCub robot for the ArmarX system. However, this unit needs to be extend to provide the correct position of the robot if the iCub is mounted on its mobile platform or a walking system is employed.

1.2.2.2 Bridging ArmarX to YARP

1.2 summarizes the interfaces (units) which have been developed to bridge ArmarX to YARP framework. These bridging components allow to completely control the ARMAR-III robot joints from the YARP framework and access its camera images in a transparent manner. Following we describe the functionality of each interface.

Component	Description
ArmarxFrameGrabber	Converting VisionX image from ArmarX to the YARP image
ArmarxMotorControl	Accessing ArmarX kinematicUnit (motor controller) from YARP
ArmarxGazeControl	Controlling ARMAR-III eyes (gaze) from YARP

Table 1.2: ArmarX to YARP bridging units.

- **ArmarXFrameGrabber**. This bridge component converts the VisionX image from ArmarX framework to the YARP image format. It registers itself to a given ArmarX image provider (e.g., the Armar-III camera), reads and converts the image from the source and stream them out through the YARP ports. Multiple YARP ports will be provided when the source frames from the ArmarX image provider contains multiple images (left and right camera images).
- **ArmarXMotorControl**. implements necessary YARP motor control interfaces to control ARMAR-III robot joints from the YARP framework. It register itself to an ArmarX kinematicUnit object and transmit the motor commands from the YARP modules to the corresponding robot or simulator. The current implementation supports controlling the ARMAR-III robot using multiple YARP interfaces such as IPositionControl, IVelocityControl, IEncoders, IControlMode and IInteractMode. Moreover, using these interfaces, the ARMAR-III robot can be directly controlled in the cartesian space using the iCub cartesian solver and controller. 1.4 demonstrates that the ARMAR-III robot joints (stimulator) are controlled using the YarpMotorGui from the YARP framework.
- **ArmarXGazeControl**. implements the gaze controller interface of the ARMAR-III robot in the YARP framework. This component bridge provides multiple gaze control functionalities in a transparent manner for the YARP modules such as looking at a fixation point and getting the 3D position of the robot eyes fixation point.

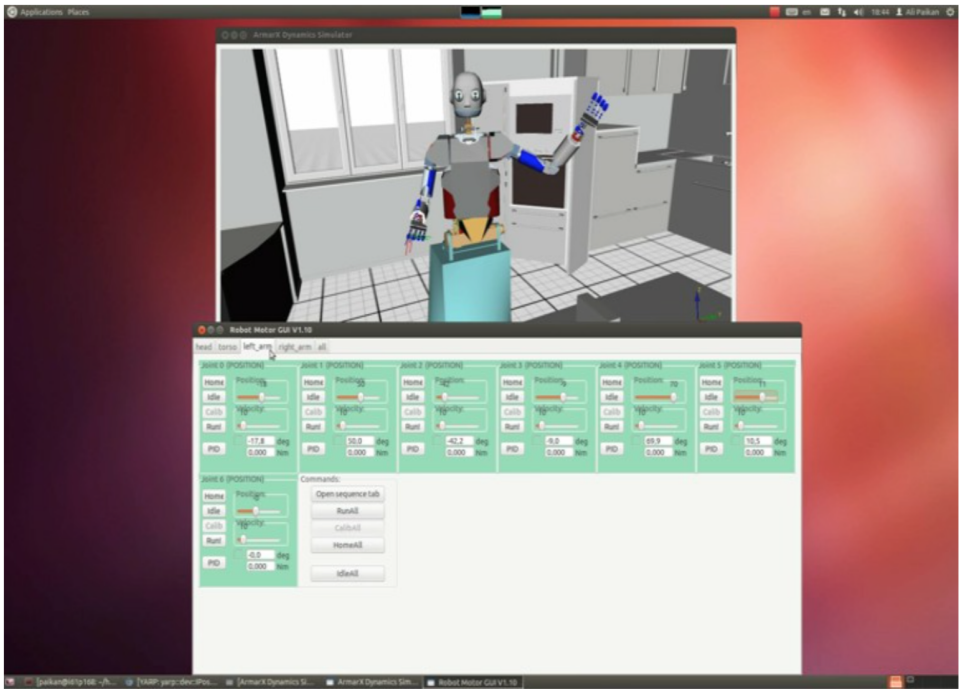


Figure 1.4: Controlling the ARMAR-III robot (simulator) using the YarpMotorGui.

1.2.2.3 Running Example

How to run cartesian controller on Armar3 robot:

```
1- run the YarpArmarxBridge
2- $ simCartesianControl --name armarx --robot armarx --context
    ArmarIIICartesianSolver --left_arm_file cartesianLeftArm.ini --
    right_arm_file cartesianRightArm.ini --no_legs
3- $ iKinCartesianSolver --context ArmarIIICartesianSolver --part left_arm
4- $ iKinCartesianSolver --context ArmarIIICartesianSolver --part right_arm
5- you can test by running cartesianInterfaceExample:
   $ cartesianInterfaceExample --robot ArmarIII --part left_arm --onlyXYZ
```

1.2.3 Xperience_LU_pipeline

Xperience_LU_pipeline is the language understanding pipeline developed at KIT that takes textual string as input and outputs a json structure containing the extracted planner goal in the PKS format, the human action description for the ELEXIR plan recognizer, the world state description to be added to the robot's memory MemoryX, the direct command representation, and some auxiliary information (e.g. object affordances); see [9] for more details. The pipeline is triggered by the ArmarX component LanguageUnderstanding that receives and further processes the resulting json structure. Xperience_LU_pipeline is based on three external software: the *Boxer* semantic parser (*http://svn.ask.it.usyd.edu.au/trac/candc/wiki/boxer*), the *Gurobi* linear solver (*www.gurobi.com*), and the Integer Linear Programming-based abductive reasoner (*https://github.com/kazeto/phillip*).

1.2.3.1 Installing Xperience_LU_pipeline

Xperience_LU_pipeline is supported for Ubuntu 14.04 and the following packages and software need to be installed.

1. Install packages

```
sudo apt-get install python2.7-dev libsqlite3-dev libpcre3-dev swi-
    prolog
```

2. Install the *Gurobi* linear solver
   (a) Go to *http://www.gurobi.com/*
   (b) Login or register your account and login
   (c) Go to DOWNLOADS→Gurobi Software→Gurobi Optimizer
   (d) Download latest Guribi version
   (e) Unpack Gurobi

```
tar xvfz gurobi*...
```

   (f) Go to DOWNLOADS→LICENSES→UNIVERSITY LICENSE
   (g) Accept license conditions and push REQUEST LICENSE, save `LICENSE_KEY` shown on the Gurobi web page
   (h) Generate the license key

```
grbgetkey LICENSE_KEY
```

   (i) Store the key (gurobi.lic) in `Xperience_LU_pipeline/gurobi/`
   (j) Export environment variables

```
1- $ export GUROBI_HOME=PATH_TO_GUROBI
2- $ export GRB_LICENSE_FILE=PATH_TO_gurobi.lic
3- $ export PATH=$PATH:$HOME/bin:$GUROBI_HOME/bin
4- $ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$GUROBI_HOME/lib\\
5- $ export LIBRARY_PATH=$LIBRARY_PATH:$GUROBI_HOME/lib\\
6- $ export CPLUS_INCLUDE_PATH=$CPLUS_INCLUDE_PATH:$GUROBI_HOME/
    include
```

3. Install the semantic parser *Boxer*

   (a) Export environment variables

```
1- $ export BOXER_DIR=DESIRED_PATH_TO_BOXER
2- $ export KIT_LU_pipeline=PATH_TO_Xperience_LU_pipeline
```

   (b) Install the SOAP server as described at *http://svn.ask.it.usyd.edu.au/trac/candc/wiki/InstallSOAP*

   (c) Register as described at *http://svn.ask.it.usyd.edu.au/trac/candc/wiki/Register* and obtain `BOXER_USERNAME` and `BOXER_PASSWORD`

   (d) Enter obtained `BOXER_USERNAME` and `BOXER_PASSWORD` in `Xperience_LU_pipeline/boxer/installation/install_boxer.sh`

   (e) Install Boxer

```
1- $ cd Xperience_LU_pipeline/boxer/installation
2- $ ./install_boxer.sh
```

4. Install abductive reasoner

   (a) Clone repository and install the reasoner

```
1- $ git clone https://github.com/kazeto/phillip
2- $ cd phillip
3- $ python tools/configure.py
4- $ make
```

   (b) Export enviroment variable

```
export PHILLIP_DIR=PATH_TO_PHILLIP
```

1.2.3.2 Running Xperience_LU_pipeline stand-alone

Xperience_LU_pipeline can be run both within ArmarX and in the stand-alone mode. To run it stand-alone, the following commands should be executed.

```
1- $ cd Xperience_LU_pipeline
2- $ ./run_Boxer_server.sh &
3- $ ./run_all.sh "INPUT TEXT"
4- $ ./kill_Boxer_server.sh
```

1.2.3.3 Running Example

```
./run_all.sh "I'd like to drink something. Bring me a multivitamin juice
    from the fridge"
```

```
{"recognize_plan": false, "commands": [], "goal_types": {"?xxu1": "
    obj_agent_human", "?x2": "obj_multivitaminjuice"}, "
    objects_talked_about": ["multivitaminjuice-n", "agent_human-n", "fridge
    -n"], "context_words": ["bring", "like", "drink"], "goals": ["(existsK
    (?xxu1 : obj_agent_human, ?x2 : obj_multivitaminjuice) K(inHandOfHuman
    (?xxu1, ?x2)))"], "SOW": [{"args": ["obj_multivitaminjuice", "
    loc_fridge"], "type": "loc", "name": "objectAt", "sign": true}], "
    human_actions": [], "feedback": []}
```


In this example, the pipeline has returned one goal for the planner (a multivitamin juice should be in a hand of human) and one world state description (a multivitamin juice is located in the fridge). No human feedback, direct commands, or human action descriptions have been detected. The pipeline has also extracted potential object affordances (*bring*, *like*, *drink*).

1.2.4 PKSplanner

The primary high level symbolic planner used on Xperience is the Planning with Knowledge and Sensing (PKS) planner which UEDIN is extending for use in the project's robotics domains. While the PKS planning system predates the Xperience project (see, e.g., [11, 10]), significant extensions were made to the planner during the project, notably involving the implementation of new application programming interfaces (APIs) for the planner to support software integration using C++ or the Internet Communications Engine (ICE) middleware (for details see the WP3.2 deliverables). A snapshot of the latest open source version of PKS has been included in the Xperience software repository. Documentation, technical papers, and software for the planner (including bugfixes), will continue to be distributed through the official PKS website at:`http://pksplanner.org/` (alternatively, `http://pks.petrick.uk/`).

1.2.5 ELEXIR

The ELEXIR plan recognition system is an open source plan recognition system that is the result of ongoing research on using probabilistic Combinatory Categorial Grammars (taken from research on natural language processing) to define and reason about plans. Papers describing the theory behind ELEXIR can be found on: `https://dl.dropboxusercontent.com/u/4326974/Site/Homepage.html`. A compressed zip file of the current version of ELEXIR with supporting documentation can be downloaded from: www.planrec.org

ELEXIR has been successfully used on the Xperience project to recognize the plans of humans in order to enable helpful interactions with humanoid robots for the joint plan of setting the table.

1.2.6 M3VR inference engine

This section introduce a maximum margin framework realizing a regression type learning in an arbitrary Hilbert space whilst the corresponding dual problem preserving the structure and, therefore, the complexity that of the binary Support Vector Machine(SVM). We provides intsructions on how the learning system can be adapted to a new task. he full system contains several modules realizing the functionality needed to provide the complete training, cross validation, prediction methods. The files implementing the training, the solver of the underlying optimization problem, the inference, prediction phase should not be modified. In the sequel those files are detailed which might be altered to adopt the system to a new learning problem. To run the system the script webrel main.py has to be started with python3. **Warning!** The modules are fully adapted to the python 3, they might not run under the python 2.

1.2.6.1 Input format

Here the Python syntax is exploited to form the data structures used. We use general variable names in the expressions, and give explanation what is the contents of those variables. The learning system assumes that the structure of the input data follows a sparse array format: *Input data: [row index array, collumn index array, values]* row index array is a integer vector of row indexes. It has to start on 0 and

has to be continuous. *collumn index array* is a integer vector of column indexes. It has to start on 0 and has to be continuous. *values* is an array of float or integer numbers. That array can have 2D structure, thus it is possible that to one row and column index belongs to a vector to be an row of values array. In the Input data only those elements of sparse array have to be enumerated whose values are available. A simple example loads the non-zero elements of the array X into the input format.

```
xdata=[[],[],[]]    ## row index, column index , values
for i in range(number_of_rows):
  for j in range(number_of_columns):
    if X[i,j]!=0:
      xdata[0].append(i)
      xdata[1].append(j)
      xdata[2].append(X[i,j])
xdata=mvm_prepare.sort_table(xdata,ifloat=1)
```

The function *mvm_prepare.sort_table(xdata,ifloat = 1)* converts the lists into arrays, sorts the elements in alphabetic order by assuming the field order *row index,column index,values*, and if the parameter *ifloat* is 1 then converts the values into floats, or if it is 0 into integers. The output of *mvm_prepare.sort_table* can be loaded into learning system:

```
xdatacls=mvm_mvm_cls.cls_mvm()   ## create the object of the learner
## ######################################################################
xdatacls.load_data(xdata,xdatarel,ncategory,nrow,ncolumn,Y0)
"""
    xdata is the output of the function mvm\_prepare.sort\_table
    xdatarel row wise additional feature vectors which can be combined
             to the basic data,
             It is [], empty list if now additional data available
    ncategory is the number of category if the "values" array contain
              categorical variables otherwise it is 0
    nrow is the number of rows
    ncolumn is the number of columns
    Y0 is an array containing the array of all possible values of
       "values", if it None then the training examples are used as possible
        values.
```

Examples can be found for data preparation in webrel load data.py and in wenrel main.py.

1.2.6.2 Setting the parameters

- **Parameters**

  The solver parameters can be found in mvm_solver_cls.py in the _init_ function. From those two are those which might be changed:

```
self.niter=200   ## maximum iteration
self.isteptype=1 ## =0 exact line serach, =1 diminishing step size
```

  If the diminishing step size is applied then the solution time is approximatelly linear in the number of iteration. The exact line search leads to a sublinear convergence, thus the execution time of the steps can increase. The diminishing step size rule significantly faster in large scale problems especially with large number of iterations.

- **Validation parameters**.

  The validation parameters can be found in mvm_validation_cls.py in the _init_ function.

```
self.vnfold=2    ## number folds(>1) in validation
self.ivalid=1    ## =0 no validation =1 validation
self.validation_rkernel=  m v m _ x    ## reference kernel to be
    validated
```

  self.validation rkernel can be overwritten, see example in webrel main.py.

- **Learning module parameters**.

  The learning module parameters can be found in mvm_mvm_cls.py in the _init_ function.

```
self.xbias=1.0  ## penalty term on bias, the deault can be =0
## combination of kernels if there more than one
self.kmode=0    ## =0 additive (feature concatenation)
                ## =1 multiplicative (fetaure tensor product)

self.crossval_mode=1  ## =0 random cross folds =1 fixtraining
self.itestmode=1  ## 0 active learning 1,2 random subsets
self.ibootstrap=2  ## =0 random =1 worst case =2 best case
                                ## =3 alternate between worst case
                                   and random
self.nrepeat=1  ## number of repetation of the folding
self.nfold=2  ## number of folds

self.nrepeat0=1  ## number of effective repetition of the folding
self.nfold0=2   ## number of effective folds
self.ieval_type=0
## type of possible output values
self.ibinary=0        ## =1 Y0=[-1,+1], =0 [0,1,...,categorymax-1]
## mvm specific
self.category=3     ## =0 rank cells =1 category cells =2 {-1,0,+1}^n
                    ## =3 joint table on all categories
## these variables are set by self.load_data method
self.categorymax=0
self.ndata=0
self.ncol=0
self.nrow=0
```

- **Kernel parameters**.

  The kernel parameters can be found in mmr_initial_params.py in the _init_ function. Here there are two collections of parameters one for the input and one for the output kernels. Each collection has

three parts, labeled with kernel, norm and cross. Within each collection one can add more than one kernels but they have to be evalueted externally.

```
parameters for output kernel - ykernel, and input kernels xkernel
    where:
                ykernel[  k e r n e l  ]  xkernel[  k e r n e l  ]
                ykernel[  n o r m  ]  xkernel[  n o r m  ]
                ykernel[  c r o s s  ]  xkernel[  c o r s s  ]
                refer to a dictionary indexed by eah of the kernels
    Parameters
    kernel parameters:  kernel_type =0 linear
                                    =1 polynomial
                                    =2 sigmoid
                                    =3 Gaussian
                        ipar1, ipar2   kernel parameters, see in
                               mvm_kernel_eval modul in kernel_nlr
                                  function
    normlization parameters:
                        ilocal : centralization
      cross(validation):
iscale : scaling parameters,
         see the coding in the mmr_normalization_new
validation for ipar1:
range (par1min,par1max) in steps par1step
 validation for ipar2:
                        range (par2min,par2max) in steps par2step
                        nrange is used in cutting the parameter range
                               by logaritmic scale for Gaussian kernel
```

- **Additional output kernel parameters** The additional output kernel parameters can be found in mmr_kernel_mvm_y.py in the _init_ function. Those parameters allow to digitalize the features created on the top of real or rank valued outputs. Those features are assumed to be Gaussian density functions with extpected value equal to the original real or rank value, and the standard deviation is taken from mmr_initial_params.py, see previous subsection.

```
self.ymax=10.0  ## Lower bounds
self.ymin=-10.0  ## upper bounds
self.yrange=20  ##number of cuting points
self.ystep=1.0  ## gap between the cutting points.
```

  See an example in webrel_main.py setting these values. Those values, except yrange are accurately fit to the observed interval in the method mvm_mvm_cls.prepare.prepare fold training. Applying small gaps, ystep, can increase the accuracy, but at the price of higher computational time.

- **Penalty parameters** The kernel parameters can be found in mmr_base_classes.py in the cls_penalty class. In the current version of the solver only Parameter C is used. To that parameter validation range can be set up within the class method set_crossval.

```
dcross={   p a r 1 m i n   : 5 ,   p a r 1 m a x   : 5,  \
                                   p a r 1 s t e p   : 1, \
                                   p a r 2 m i n   : 0.0 ,   p a r 2 m a x   :
                                     0.0, \
                                   p a r 2 s t e p   : 1, \
                                   n r a n g e   : 0 }
```

  The validation range is given for C by [par1min, par1max] with step size par1step.

- **Normalization parameters** The normalization parameters can be found in mmr_normalization_new.py.

```
## function to normalize the input and the output data
## !!!! the localization happens before normalization if both given
    !!!
## input
##   ilocal centralization
##      =-1 no localization
##      =0 mean
##      =1 median
##      =2 geometric median
##      =3 shift by ipar
##      =4 smallest enclosing ball
##      =5 row mean row wise
## icenter
##      =-1 no scaling
##      =0 scale item wise by L2 norm
##      =1 scale item wise by L1 norm
##      =2 scale item wise by L_infty norm
##      =3 scale items by stereographic projection relative to zero
##      =4 scale variables by STD(standard deviation)
##      =5 scale variables by MAD(median absolute deviation)
##      =6 scale variables by absolute deviation
##      =7 scale all variables by average STD
##      =8 scale all variables by maximum STD
##      =9 scale all variables by median MAD
##      =10 scale item wise by Minkowski norm, power given by ipar
##      =11 \sum_i||u-x_i||/m where u=0
##      =12 scale all variables by overall max
##      =13 Mahalonobis scaling
## XTrain   Data matrix which will be normalized. It assumed the
##                      rows are the sample vetors and the columns are
     variables
##   XTest  Data matrix which will be normalized. It assumed the
##                  rows are the sample vetors and the columns are
    variables.
##                      It herites the center and the scale in the
    variable
##                      wise case from the XTrain, otherwise it is
    normalized
##                      independently
##   ipar       additional parameter
##
##  output
##   XTrain Data matrix which is the result of the normalization
##                      of input XTrain. It assumed the rows are the
    sample
##                      vetors and the columns are variables
## XTest        Data matrix which is the result of the normalization
##                      of input XTest. It assumed the rows are the
    sample
##                      vetors and the columns are variables.
## opar         the radius in case of ixnorm=2.
```

1.2.7 RobWork

RobWork is a collection of software package for simulation and control of robot systems and is used for research and education as well as for practical robot applications. The software is comprised of a core package and three optional odd-ons. The core contains basic features such as kinematic modelling, path planning, simulation of sensors and task/trajectory representations. The add-ons denoted RobWorkStudio, RobWorkHardware and RobWorkSim, provides a graphical user interface, a collection of hardware drivers that connect into RobWork and support for rigid body simulation used for grasp and assembly simulation.

During the Xperience project significant extensions have been made on the simulation side to support the Xperience use cases. The main new components are a model for additional object types and the more realistic simulation of Kinect cameras with artificial scenes. RobWork was used as a basis for SDU's work on grasping unmodelled objects ([13, 14]) and the whole work on learning visual procondistions (e.g., [2]) as well as for all practical robotics setups done within Xperience at SDU.

An extensive documentation with information on how to install and use RobWork can be found at: `http://www.robwork.dk/`.

1.2.8 Covis

Similar to RobWork, Covis provides a C++ library of facilities, but in this case for solving computer vision tasks, e.g. related to robotic contexts. Covis contains functionality for solving low-level computer vision problems, such as basic 2D and 3D processing, as well as high-level functionality, such as pose estimation and object recognition. One of the main design ideas behind Covis is that it should expose functionalities at different levels. At the highest level, algorithms can be applied by a simple "one-shot" function calls with few parameters required. This makes the available algorithms very accessible to new users. On the other hand, if the programmer wishes to modify more detailed aspects of an algorithm, Covis also exposes all the underlying parameters at a lower level such that the applied algorithm can be tuned to specific use cases.

During the Xperience project Covis was started based on an old version with similar properties (called CoViS). Functinality in Covis made for Xperience is besides others used in our work on learning visual procondistions (e.g., [2]) and in the work on suggesting affordance similar objects based on visual apperance ([6, 7]).

Covis can be found at: `https://gitlab.com/caro-sdu/covis` with sample applications at: `https://gitlab.com/caro-sdu/covis-app`.

1.2.9 Motor action replacement and DMP learning package

Here we introduce the Matlab code used for motor action replacement and DMP learning, which have been tested developed and tested at JSI using KUKA LWR robot. Please refer to deliverable **D1.2.2**, Chapter 2.1.4 and references [16, 8] for additional information on the action replacement mechanism. This software is available in the Xperience repository at:

`http://svn.code.sf.net/p/xperience/code/trunk/main/src/ActionReplacement/`

1.2.9.1 DMP and AL-DMP learning

We first describe the code that implements standard discrete and periodic Dynamic Movement Primitives (DMPs) [3] and Arc Length Dynamic Movement Primitives (AL-DMPs) [15]. Standard DMPs are used in most of our motor action adaptation and control experiments, whereas AL-DMPs are suitable for action recognition and statistical generalization.

For each DMP type, there exist two basic functions: a training function, which trains DMP parameters given a trajectory, and the integration function, which does the opposite, i. e. reconstructs the trajectory given the DMP system. The usage of these functions is illustrated in test scripts attached to the modules. The available functions are summarized in Table 1.3 and 1.4.

dmpLib	
dmpLib/DMP_train.m	trains a discrete DMP given trajectory y
dmpLib/DMP_integrate.m	integrates one sample of a discrete DMP given the current DMP state
dmpLib/test_case_dmp.m	runs the training and integration functions on an example 2 DOF trajectory
dmpLib/PDMP_train.m	trains a periodic DMP given periodic signal y
dmpLib/PDMP_integrate.m	integrates one sample of a periodic DMP given the current DMP state
dmpLib/test_case_pdmp.m	runs the training and integration functions on an example 2 DOF trajectory

Table 1.3: Summary of DMP training and reconstruction code

aldmpLib	
aldmpLib/ALDMP_train.m	trains an AL-DMP representation given a trajectory
aldmpLib/ALDMP_reconstruct.m	reproduces a trajectory given an AL-DMP
aldmpLib/ALDMP_integrate.m	integrates one sample of the DMP
aldmpLib/test_case_aldmp.m	runs the training and reconstruction functions on an example 2 DOF trajectory
aldmpLib/interparc.m	interpolates a given trajectories at equdistant points, needed by ALDMP_train

Table 1.4: Summary of AL-DMP training and reconstruction code

1.2.9.2 Motor Action Replacement

The core code for action replacement is implemented in two functions: `ActionTransfer_init.m` and `ActionTransfer.m`. The first function is used for initialization purposes: definition of a DMP describing the motor action that needs to be replaced, initialization of robot parameters, and initialization of learning constants and parameters required by the main action replacement function `ActionTransfer.m`. `ActionTransfer.m` adapts the robot motion in order to minimize the error between the desired reference signal and the actual signal sensed by the robot during action execution. The ILC (iterative learning control) framework is used for the adaptation of motor commands.

The reference signal for action transfer is case specific. For example, in the case of replacement of a wiping DMP by a stirring DMP (see [16]), the reference signal can be defined by the desired forces that should arise during action execution, as provided in function `GetRefSignal.m`. In simulation another function `GetActSignal.m` is needed, which calculates the corresponding signal that is sensed by a robot

lwrLib	
Init(arm,mode,tsamp,env)	initializes Matlab - LWR interface, arm (left, right), control mode (1 : stiff joint, 2 : Cartesian compliance, 3 : joint compliance), env : name of the simulation environment, if omitted, initializes real robot environment
SetTCP(T)	sets tool center point, T is tcp homogenous matrix
T = MakeT(R,p)	makes homogenous transformation form rotation matrix R and position vector p
Jmove(q,time)	moves arm to joint configuration q in time seconds using joint interpolation
SetStrategy(mode)	selects LWR control mode, see Init
CmoveFor(dp, time)	displaces current pose for dp = [x y z] in time seconds using Cartesian interpolation
T = RobotT('m')	reads the measured robot pose in homogeneous transformation
Cmove(T,time)	moves the arm to the pose T in time seconds using Cartesian interpolation
Dmove(T)	moves arm to the pose T without interpolation
Close	closes Matlab - LWR interface

Table 1.5: Functions needed to control the robot during action replacement learning.

during action execution. In the example function `GetActSignal.m`, forces on the robot tcp during the action execution in a simulated environment are calculated.

A virtually identical code was used to run real robot experiments. The only modification was that we linked the library `lwrLib` and provided additional parameters in initialization function `Init` in `ActionTransfer_init.m`.

The `ActionTransfer_init.m` and `ActionTransfer.m` use the following libraries:

- dmpLib - for DMP trajectory encoding and integration, applied to both discrete and periodic signals. This library is attached to this deliverable.
- lwrLib - library for driving Kuka LWR robot using FRI (Fast Robot Interface) protocol. There are two versions of the library, one for the real environment and one for the simulated environment, which uses RoboWorks engine (`http://newtonium.com/`) for graphical representation. The library for working with simulated environment is attached to this deliverable.

Further explanation of commands from external libraries, which are used used in `ActionTransferDemo.m`, is provided in Table 1.5.

The test program is `ActionTransferDemo.m`. The provided code can be used to demonstrate action replacement in a simulated environment.

1.2.10 YARP and ICUB Software

YARP stands for Yet Another Robot Platform, an open-source project that encapsulates lessons from our experience in building humanoid robots. The goal of YARP is to minimize the effort devoted to infrastructure-level software development by facilitating code reuse, modularity and so maximize research-level development and collaboration. Humanoid robotics is a bleeding edge field of research, with constant flux in sensors, actuators, and processors. In brief, YARP provides an intercommunication layer that allows processes running on different machines to exchange data travelling through named connection nodes called ports. Communication is platform and transport independent: processes are not aware of the details of the underlying operating system or protocol and can be relocated at will across the available machines on a network. More importantly, since connections are established at runtime, it is easy to dynamically modify how data travels across processes, as well as addition of new modules or removal of existing ones. Interface between modules is specified in terms of YARP ports (i.e. port names) and the type of data these ports receive or send (respectively for input or output ports). This modular approach allows minimizing the dependency between algorithm and the underlying hardware/robot; different hardware devices become interchangeable as long as they export the same interface. Finally, YARP is written in C++, so it is normally used as a library in C++ code. However, any application that has a TCP/IP interface can talk to YARP modules using a standard data format. YARP is currently used and tested on Windows, Linux and OSX which are common operating systems used in robotics. The YARP software can be found in the YARP GIT repository and downloaded from:

- **YARP**: `https://github.com/robotology/yarp.git`

More specifically, YARP supports building a robot control system as a collection of programs communicating in a peer-to-peer way, with an extensible family of connection types (tcp, udp, multicast, local, MPI, mjpg-over-http, XML/RPC, tcpros, ...) that can be swapped in and out to match your needs. We also support similarly flexible interfacing with hardware devices. Our strategic goal is to increase the longevity of robot software projects.

The software on the iCub is built based on modules which communicate with each other using YARP, enabling multi-machine and multi-platform integration. The iCub repository contains all the low-level (firmware+controllers) and high level software (cognitive architectures based on motoric, visual and speech pipelines). These can be found in the iCub GIT repository and downloaded from:

- **ICUB**: `https://github.com/robotology/icub-main.git`

1.2.11 Affordance Learning Code

The software related to the project on affordances and robotic tool use is divided in two blocks. The first of them corresponds to the software that runs online on the robot, thus written in C++, and is in charge of the motor control and perception of the robot performed during the experimental procedures. The second block deals with the learning part and experimental data analysis, and is written in MATLAB for ease and better visualization of results. Let's see them in detail.

The first block contains the methods to run the experiments described in [5] (ICRA) and [4] (Humanoids). These are organized in a series of YARP modules and applications which basically perform action execution, tool feature extraction and affordance values measurement, as well as coordination and communication among all the different modules. The modules specifically implemented for the current project on tool use affordances can be found in two separate repositories: https://github.com/robotology/affordances and `https://github.com/robotology/affordances/objects3DModeler`. The reason for this separation is that the 3D processing performed on the second experiment [4] relies heavily on the Point Cloud Library (PCL), while the rest of the modules have ICUB and YARP dependencies (see 1.2.10). Therefore, the repository objects3DModeler was set to contain those modules that deal directly with 3D, freeing the affordances module from the dependency on PCL. Nevertheless, both experiments relied also on many modules that are available at the main **YARP** and **iCub** large software repository (`https://github.com/robotology`) for action execution (KARMA, IOL and all the robot's low level communication) and perception pipeline (templatePFTracker to track target object, lbpExtract to segment all objects).

The second block consists of the methods implemented in MATLAB in order to perform the offline part of the data processing required for the aforementioned experiments. This includes, on the one hand, the learning procedures carried out with the data gathered from performing the experiments, including clustering as well as model training, and on the other hand, the methods required to test the models and evaluate the yielded results as well as to generate the visualizations used to interpret them. This MATLAB code can be found on the affordances repository, under the MATLAB subfolder: `https://github.com/robotology/affordances/tree/master/MATLAB`

1.2.12 LCCP

LCCP aims at segmenting a point cloud of a scene into objects using a purely data-driven method. This produces results that are comparable to state-of-the-art methods which incorporate high-level concepts involving classification, learning and model fitting. The algorithm begins by decomposing the scene into an adjacency-graph of surface patches based on a voxel grid. Edges in the graph are then classified as either convex or concave using a combination of simple criteria which operate on the local geometry of these patches. This way the graph is divided into locally convex connected subgraphs, which with high accuracy represent objects. Additionally, it uses a depth dependent voxel grid (-tvoxel parameter of the example program) to deal with the decreasing point-density at far distances in the point clouds. This improves segmentation, allowing the use of fixed parameters for vastly different scenes. The algorithm has been implemented within the SEGMENTATION module in the Point Cloud Library (PCL).

1.2.13 Planning with object substitution (POS)

The Matlab toolbox for planning with object substitution (POS) contains functions for finding replacement of objects considered in a plan but missing from the current scenario. Plan generation is carried out from prototypical planning problem definitions (like recipes in a cooking book). Once a plan is generated, the system evaluates if all the objects considered in the plan are present in the scenario. In case of missing objects, the system extracts the affordances of the missing objects from the predicates in the precondition part of the planning operators involved in the plan and searches for objects in the scenario having the same affordances of the missing objects. The affordances of the objects in the scenario are extracted from the ROAR repository ([12]). The access to the ROAR tables is carried out using a mex function also included in the toolbox. If the systems finds objects in the scenario having the same affordances, it uses these objects to replace the missing ones. For a thorough description of the algorithms implemented in the POS toolbox, please refer to [1].

References

[1] A. Agostini, M.J. Aein, S. Szedmak, E.E. Aksoy, J. Piater, and F. Wörgötter. Using Structural Bootstrapping for Object Substitution in Robotic Executions of Human-like Manipulation Tasks. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 6479–6486, 2015.

[2] Severin Fichtl, Dirk Kraft, Norbert Krüger, and Frank Guerin. Using relational histogram features and action labelled data to learn preconditions for means-end actions. In *IROS 2015 workshop on sensorimotor contingencies for robotics*, 2015.

[3] A. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural Computation*, 25(2):328–373, 2013.

[4] Tanis Mar, Vadim Tikhanoff, Giorgio Metta, and Lorenzo Natale. Multi-model approach based on 3d functional features for tool affordance learning in robotics. In *Humanoid Robots (Humanoids), 2015 15th IEEE-RAS International Conference on*, 2015.

[5] Tanis Mar, Vadim Tikhanoff, Giorgio Metta, and Lorenzo Natale. Self-supervised learning of grasp dependent tool affordances on the icub humanoid robot. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3200–3206. IEEE, 2015.

[6] Wail Mustafa, Dirk Kraft, and Norbert Krüger. Extracting categories by hierarchical clustering using global relational features. In Roberto Paredes, Jaime S. Cardoso, and Xosé M. Pardo, editors, *Pattern Recognition and Image Analysis*, volume 9117 of *Lecture Notes in Computer Science*, pages 541–551. Springer International Publishing, 2015.

[7] 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.

[8] B. Nemec, T. Petrič, and A. Ude. Force adaptation with recursive regression iterative learning controller. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2835–2841, Hamburg, Germany, 2015.

[9] Ekaterina Ovchinnikova, Mirko Wachter, Valerij Wittenbeck, and Tamim Asfour. Multi-purpose natural language understanding linked to sensorimotor experience in humanoid robots. In *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference*, pages 365–372. IEEE, 2015.

[10] Ronald P. A. Petrick and Fahiem 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, Menlo Park, CA, April 2002. AAAI Press.

[11] Ronald P. A. Petrick and Fahiem Bacchus. Extending the knowledge-based approach to planning with incomplete information and sensing. In Shlomo Zilberstein, Jana Koehler, and Sven Koenig, editors, *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pages 2–11, Menlo Park, CA, June 2004. AAAI Press.

[12] Sandor Szedmak, Emre Ugur, and Justus Piater. Knowledge Propagation and Relation Learning for Predicting Action Effects. pages 623–629. IEEE, 09 2014.

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

[14] Mikkel Tang Thomsen, Dirk Kraft, and Norbert Krüger. A semi-local surface feature for learning successful grasping affordances. In *VISAPP International Conference on Computer Vision Theory and Applications*, 2016. (accepted).

[15] A. Ude, R. Vuga, B. Nemec, and J. Morimoto. Movement representation with dynamic movement primitives parameterized by arc length. In *Submitted to IROS*, 2016.

[16] F. Wörgötter, C. Geib, M. Tamosiunaite, E. E. Aksoy, J. Piater, H. Xiong, A. Ude, B. Nemec, D. Kraft, N. Krüger, M. Wächter, and T. Asfour. Structural bootstrapping – a novel, generative mechanism for faster and more efficient acquisition of action-knowledge. *IEEE Transactions on Autonomous Mental Development*, 7(2):140–154, 2015.