Knowledge Processing for Autonomous Robots
Knowing where to look for something

- **Naive:**
  Fixed controller for opening everything with a handle
Knowing where to look for something

- Naive:
  Fixed controller for opening everything with a handle

- Parameterizable:
  Look up the positions of all containers in the map
Knowing where to look for something

- **Naive:**
  Fixed controller for opening everything with a handle

- **Parameterizable:**
  Look up the positions of all containers in the map

- **Knowledge-based:**
  Infer the right container based on the types of the object and the container
Knowing where to look for something
Knowledge-based decision making

Robot control program

```plaintext
routine fetch (obj)
    loc ← likely-storage-loc(obj)
    if in-container(loc, container)
    then
        traj ← articulation(container)
        open-container(container, traj)
        pick-up(obj)
```

Formulate control decisions as inference tasks
Separate control flow from execution context
Increase re-usability of robot plans
CRAM: Cognitive Robot Abstract Machine
Outline

- **Wednesday:** Knowledge representation (KnowRob, ProbCog)
- **Thursday morning:** CRAM execution framework
- **Thursday afternoon:** Semantic perception methods
- **Friday:** Group discussions about specific project ideas
Outline: KnowRob tutorial

- **Introduction**: What can CRAM, KnowRob, etc be used for?
- **Languages**: Prolog, OWL, Java, etc
- **Basics**: launching KnowRob, loading files, querying, debugging
- **Objects**: class taxonomy, instances, components, environment models, units and coordinates, qualitative spatial relations, computable properties
- **Actions**: class-level descriptions, properties, projection, planning, action dependencies
- **RoboEarth**: concept, interface to KnowRob, up- and download of information, capability matching
- **ProbCog**: interface to statistical relation inference (Dominik)
What I won’t talk much about...

- **Time**: Reasoning about time points and intervals, object poses that change over time

- **Changing objects**: Creation/destruction/modification of objects through actions and processes

- **Interfaces to external components**: Perceptual methods, SQL databases, Web information

- **Human activity data**: segmentation/classification/abstraction/reasoning about observations of human activities

- **Machine learning**: clustering and classification using Weka and Mallet libraries
KnowRob components

Knowledge processing system

Knowledge acquisition from the WWW

Representations for robot knowledge

Observations of human actions
The KnowRob knowledge processing system
KnowRob: A practical robot knowledge base

Reasoning methods
- Description logics inference
- ProbCog probabilistic inference
- Computable classes and properties
- Robot capability matching
- Classification and clustering methods
- Semantic similarity measures

Knowledge acquisition
- Web instructions
- Online shops
- Observations of humans

Integration with the robot
- Robot middleware
- json_prolog query interface
- Vision system

Interaction with humans
- Visualization modules
- Dialog module and speech interface

[IIROS 2009, RAS 2010]
Representations for robot knowledge
Semantic object and environment models

- Recognized objects and environment map represented in the knowledge base
- Pieces of furniture as object instances that inherit properties of their types
- Articulation models for opening containers
- Spatio-temporal representation of object poses

Image courtesy of Ulrich Klank
Actions and processes

- Actions move, split, destroy, create, join, switch on, open and close objects
- Processes are started as indirect effects of actions
- Integrated representation to jointly reason about actions and processes
- Prediction of their effects using qualitative high-level projection rules
Composing actions to plans

ServeADrink
dependsOnComponent ObjectRecognitionModel AND providesModelFor.Bottle
dependsOnComponent ObjectRecognitionModel AND providesModelFor.Bed
dependsOnComponent ObjectRecognitionModel AND providesModelFor.Cabinet

MoveBaseToGraspPose
subClassOf Translation toLocation (Point2D AND inReachOf.bottle1)

GraspBottle
subClassOf GraspingSomething objectActedOn bottle1

MoveBaseToHandoverPose
subClassOf Translation toLocation (Point2D AND inReachOf.bed1)

HandoverObject
subClassOf Reaching toLocation (Point3D AND aboveOf.bed1)

OpenGripperForHandover
subClassOf OpeningAGripper deviceUsed robotRightHand

GraspBottle
subClassOf GraspingSomething objectActedOn bottle1

ReachToApproachPose
subClassOf Reaching handPose-approachObject toLocation

OpenGripperForGrasping
subClassOf OpeningAGripper handPose-approachObject handPose-graspObject toLocation

ReachForObjectPose
subClassOf Reaching handPose-graspObject toLocation

CloseGripperForGrasping
subClassOf ClosingAGripper handPose-graspObject toLocation

LiftObjectToApproachPose
subClassOf Reaching handPose-approachObject toLocation

ReachToParkingPose
subClassOf Reaching parkingPose toLocation
Web-enabled knowledge bases
Task instructions from the WWW

Semantic parsing

WordNet lexical database

OpenCyc ontology

Word sense resolution

Ontology mapping

Robot plan generation

place/ N08664443, V01494310, V02392762, ...

cup/ N03147509, N13619168, ...

table/ N08266235, N04379243

place/ PuttingSthSomewhere
cup/ DrinkingMug
table/ Table-PieceOfFurniture
Object models

- Automatically created ontology of >7500 objects from the online shop germandeli.com
- Class hierarchy from categories + perishability, weight, price, origin, ...
- SIFT recognition models from product pictures (work by Dejan Pangercic)
Understanding observations of human actions
Acquiring knowledge by observing humans

- Gather knowledge from observing human activities
- Interpret the data to obtain reusable information

Image courtesy of Jan Bandouch
Segmentation and abstraction of motions

- Conditional Random Fields-based motion segmentation
- Pose-related features and data from external sensors
Extracting knowledge by observing humans
Integrated experiments
Completing underspecified instructions

take egg and milk from fridge → crack the egg → mix flour and milk → mix the egg yolk with the dough → pour the dough onto the pancake maker → baking → flip the pancake → baking

- Action sequence
- Action effects
- Object models
- Environment information
- Observations of humans
- Process models
Comparing observations with WWW instructions

TransportPlaceMat
- subClassOf: TransportingSth
- objActedOn: PlaceMat
toLocation: inFrontOfChair

TransportNapkin
- subClassOf: TransportingSth
- objActedOn: Napkin
toLocation: rightOfPlaceMat

TransportDinnerPlate
- subClassOf: TransportingSth
- objActedOn: DinnerPlate
toLocation: centerOfPlaceMat

TransportTableKnife
- subClassOf: TransportingSth
- objActedOn: TableKnife
toLocation: rightOfDinnerPlate

TransportCup
- subClassOf: TransportingSth
- objActedOn: PlaceMat
toLocation: rightOfDinnerPlate

match & compare

move  transport  move  open cupboard  move
move  reach  take  move  put  release  move  reach  open door  close door  release  move

TUM}

robohow
Inferring missing objects

- Given: incomplete table setup
- Objective: infer which objects are additionally needed
- Integrates perception with statistical relational models of human preferences

(see also comp_missingobj, prolog_perception, mod_probcog)
Outline: KnowRob tutorial

- **Introduction**: What can CRAM, KnowRob, etc be used for?
- **Languages**: Prolog, OWL, Java, etc
- **Basics**: launching KnowRob, loading files, querying, debugging
- **Objects**: class taxonomy, instances, components, environment models, units and coordinates, qualitative spatial relations, computable properties
- **Actions**: class-level descriptions, properties, projection, planning, action dependencies
- **RoboEarth**: concept, interface to KnowRob, up- and download of information, capability matching
- **ProbCog**: interface to statistical relation inference (Dominik)
Why Prolog for autonomous robots?

- Can be read declaratively
- Can be programmed (prodecurally)
- There are textbooks and tutorials
- Industrial-strength open-source implementations
- Foreign language interfaces
- Result interpretation: similar to a DB query language

KnowRob: based on SWI Prolog (http://www.swi-prolog.org)
Which language for what purpose?

**OWL:**
- Class taxonomy of objects, actions, events,...
- Instances of these classes (e.g. environment models, experiences)
- Robot capabilities/action requirements

**Prolog:**
- Internal representation (OWL parsed into Prolog triples)
- Query language
- Inference predicates (DL inference, computables,...)

**Java:**
- External interfaces (WWW, ROS,...)
- Library integration (ProbCog, Weka,...)
Prolog syntax

- Predicate names and constants start with a lowercase letter or are inside single quotes
- Variables start with an uppercase letter
- Formulas end with a full stop ‘.’
- Stepping through all results of a query with ‘;’
- Logical operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Prolog equivalent</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>∧</td>
<td>,</td>
<td>type(A, 'Cup'), on(A, T).</td>
</tr>
<tr>
<td>∨</td>
<td>;</td>
<td>type(A, 'Food'); type(A, 'Drink').</td>
</tr>
</tbody>
</table>

Example

```prolog
dairyproduct(milk).
perishable(Stuff) :- storedIn(Stuff, Ref), refrigerator(Ref).
```
**Introduction to Prolog**

`milk.pl`

```prolog
cup(cup0).
dairyProduct(milk1).
meatProduct(ham2).

cupboard(cupboard3).
refrigerator(fridge4).

perishable(Prod) :- dairyProduct(Prod); meatProduct(Prod).

storagePlaceFor(Loc, Item) :- refrigerator(Loc), perishable(Item).
storagePlaceFor(Loc, Item) :- cupboard(Loc), cup(Item).

searchForIn(Item,Loc) :- storagePlaceFor(Loc, Item).
```
Example queries

Two main types of queries:

- Query with unbound variables $\rightarrow$ variable bindings
- All variables bound $\rightarrow$ true/false result

```
$ prolog -f milk.pl

?- refrigerator(A).
A = fridge4.

?- refrigerator(fridge4).
true.

?- searchForIn(milk1, A).
A = fridge4.
```
Example queries

?- storagePlaceFor(A, milk1).
A = fridge4 .

?- storagePlaceFor(fridge4, A).
A = milk1 ;
A = ham2 ;

?- storagePlaceFor(A, B).
A = fridge4,
B = milk1 ;
A = fridge4,
B = ham2 ;
A = cupboard3,
B = cup0.
Under the hood: the tracer

Tracing a query

?- trace, searchForIn(milk1, P).
Call: (7) searchForIn(milk1, _G369) ? creep

milk.pl

searchForIn(Item, Loc) :- storagePlaceFor(Loc, Item).
Under the hood: the tracer

Tracing a query

?- trace, searchForIn(milk1, P).
  Call: (7) searchForIn(milk1, _G369) ? creep
  . Call: (8) storagePlaceFor(_G369, milk1) ? creep

milk.pl

storagePlaceFor(Loc, Item) :- refrigerator(Loc), perishable(Item).
storagePlaceFor(Loc, Item) :- cupboard(Loc), cup(Item).
Tracing a query

?- trace, searchForIn(milk1, P).
  Call: (7) searchForIn(milk1, _G369) ? creep
  .  Call: (8) storagePlaceFor(_G369, milk1) ? creep
  .  .  Call: (9) refrigerator(_G369) ? creep

milk.pl

refrigerator(fridge4).
Under the hood: the tracer

Tracing a query

?- trace, searchForIn(milk1, P).
   Call: (7) searchForIn(milk1, _G369) ? creep
      . Call: (8) storagePlaceFor(_G369, milk1) ? creep
      . . Call: (9) refrigerator(_G369) ? creep
      . . Exit: (9) refrigerator(fridge4) ? creep

milk.pl

refrigerator(fridge4).
Under the hood: the tracer

Tracing a query

```prolog
?- trace, searchForIn(milk1, P).
  Call: (7) searchForIn(milk1, _G369) ? creep
  .  Call: (8) storagePlaceFor(_G369, milk1) ? creep
  .  .  Call: (9) refrigerator(_G369) ? creep
  .  .  Exit: (9) refrigerator(fridge4) ? creep
  .  .  Call: (9) perishable(milk1) ? creep
```

milk.pl

```prolog
dairyProduct(milk1).
perishable(Prod) :- dairyProduct(Prod); meatProduct(Prod).
```
Under the hood: the tracer

Tracing a query

?- trace, searchForIn(milk1, P).
  Call: (7) searchForIn(milk1, _G369) ? creep
  .  Call: (8) storagePlaceFor(_G369, milk1) ? creep
  .  .  Call: (9) refrigerator(_G369) ? creep
  .  .  Exit: (9) refrigerator(fridge4) ? creep
  .  .  Call: (9) perishable(milk1) ? creep
  .  .  .  Call: (10) dairyProduct(milk1) ? creep

milk.pl

dairyProduct(milk1).
perishable(Prod) :- dairyProduct(Prod); meatProduct(Prod).
Under the hood: the tracer

Tracing a query

?- trace, searchForIn(milk1, P).
    Call: (7) searchForIn(milk1, _G369) ? creep
        . Call: (8) storagePlaceFor(_G369, milk1) ? creep
        . . Call: (9) refrigerator(_G369) ? creep
        . . Exit: (9) refrigerator(fridge4) ? creep
        . . Call: (9) perishable(milk1) ? creep
        . . . Call: (10) dairyProduct(milk1) ? creep
        . . . Exit: (10) dairyProduct(milk1) ? creep

milk.pl

dairyProduct(milk1).
perishable(Prod) :- dairyProduct(Prod); meatProduct(Prod).
Under the hood: the tracer

Tracing a query

?- trace, searchForIn(milk1, P).
  Call: (7) searchForIn(milk1, _G369) ? creep
   .  Call: (8) storagePlaceFor(_G369, milk1) ? creep
   .  .  Call: (9) refrigerator(_G369) ? creep
   .  .  Exit: (9) refrigerator(fridge4) ? creep
   .  .  Call: (9) perishable(milk1) ? creep
   .  .  .  Call: (10) dairyProduct(milk1) ? creep
   .  .  .  Exit: (10) dairyProduct(milk1) ? creep
   .  .  Exit: (9) perishable(milk1) ? creep

milk.pl

dairyProduct(milk1).
perishable(Prod) :- dairyProduct(Prod); meatProduct(Prod).
Under the hood: the tracer

Tracing a query

?- trace, searchForIn(milk1, P).
  Call: (7) searchForIn(milk1, _G369) ? creep
  .  Call: (8) storagePlaceFor(_G369, milk1) ? creep
  .  .  Call: (9) refrigerator(_G369) ? creep
  .  .  Exit: (9) refrigerator(fridge4) ? creep
  .  .  Call: (9) perishable(milk1) ? creep
  .  .  .  Call: (10) dairyProduct(milk1) ? creep
  .  .  .  Exit: (10) dairyProduct(milk1) ? creep
  .  .  Exit: (9) perishable(milk1) ? creep
  .  Exit: (8) storagePlaceFor(fridge4, milk1) ? creep

milk.pl

storagePlaceFor(Loc, Item) :- refrigerator(Loc), perishable(Item).
storagePlaceFor(Loc, Item) :- cupboard(Loc), cup(Item).
Under the hood: the tracer

Tracing a query

?- trace, searchForIn(milk1, P).
   Call: (7) searchForIn(milk1, _G369) ? creep
   .   Call: (8) storagePlaceFor(_G369, milk1) ? creep
   .   .  Call: (9) refrigerator(_G369) ? creep
   .   .  Exit: (9) refrigerator(fridge4) ? creep
   .   .  Call: (9) perishable(milk1) ? creep
   .   .  .  Call: (10) dairyProduct(milk1) ? creep
   .   .  .  Exit: (10) dairyProduct(milk1) ? creep
   .   .  Exit: (9) perishable(milk1) ? creep
   .   Exit: (8) storagePlaceFor(fridge4, milk1) ? creep
Exit: (7) searchForIn(milk1, fridge4) ? creep

milk.pl

searchForIn(Item, Loc) :- storagePlaceFor(Loc, Item).
Tracing a query

?- trace, searchForIn(milk1, P).
   Call: (7) searchForIn(milk1, _G369) ? creep
     . Call: (8) storagePlaceFor(_G369, milk1) ? creep
       . . Call: (9) refrigerator(_G369) ? creep
       . . Exit: (9) refrigerator(fridge4) ? creep
       . . Call: (9) perishable(milk1) ? creep
       . . . Call: (10) dairyProduct(milk1) ? creep
       . . . Exit: (10) dairyProduct(milk1) ? creep
       . . Exit: (9) perishable(milk1) ? creep
       . Exit: (8) storagePlaceFor(fridge4, milk1) ? creep
   Exit: (7) searchForIn(milk1, fridge4) ? creep
P = fridge4 .

milk.pl

searchForIn(Item, Loc) :- storagePlaceFor(Loc, Item).
Why is this useful for robotics?

- Separation of the inference task (what to infer) from the implementation (how to infer it) and the data (what evidence to infer it from)

- Search-based inference: easy integration of additional inference methods or knowledge sources as additional branches in the search tree
Outline: KnowRob tutorial

- **Introduction**: What can CRAM, KnowRob, etc be used for?
- **Languages**: Prolog, OWL, Java, etc
- **Basics**: launching KnowRob, loading files, querying, debugging
- **Objects**: class taxonomy, instances, components, environment models, units and coordinates, qualitative spatial relations, computable properties
- **Actions**: class-level descriptions, properties, projection, planning, action dependencies
- **RoboEarth**: concept, interface to KnowRob, up- and download of information, capability matching
- **ProbCog**: interface to statistical relation inference (Dominik)
Launching KnowRob

KnowRob is very modular: Dependencies are automatically loaded by rosprolog

General syntax: `rosrun rosprolog rosprolog <pkg>`
- starts a Prolog environment
- checks for dependencies of `<pkg>`
- locates them on the disk
- loads the prolog/init.pl file for each of these packages

For today we use:
`rosrun rosprolog rosprolog knowrob_tutorial`
The TBOX contains **terminological statements**: abstract descriptions of classes and properties.

- `owl_subclass_of(Sub, Super)`. reads all classes derived from Super.
- `rdfs_subproperty_of(SubProp, Prop)`. lists sub-properties of Prop.
- `owl_individual_of(Restr, owl:'Restriction'), owl_subclass_of(Class, Restr)`. reads all restrictions that are defined on Class.
- `owl_satisfies_restriction(Res, Restr)`. verifies whether Res satisfies the restriction Restr.
- `owl_description(RestrOWL, RestrProlog)`. translates the OWL identifier of a restriction into a Prolog term.

(see also [http://www.swi-prolog.org/pldoc/package/semweb.html](http://www.swi-prolog.org/pldoc/package/semweb.html))
Example queries: TBOX

$ rosrun rosprolog rosprolog knowrob_tutorial

?- owl_subclass_of(A, knowrob:'FoodOrDrink').
A = 'http://ias.cs.tum.edu/kb/knowrob.owl#FoodOrDrink' ;
A = 'http://ias.cs.tum.edu/kb/knowrob.owl#Drink' ;
A = 'http://ias.cs.tum.edu/kb/knowrob.owl#Coffee-Beverage' ;
A = 'http://ias.cs.tum.edu/kb/knowrob.owl#InfusionDrink' ;
A = 'http://ias.cs.tum.edu/kb/knowrob.owl#Tea-Beverage' ;
A = 'http://ias.cs.tum.edu/kb/knowrob.owl#Tea-Iced'
ABOX queries

- The ABOX contains assertional statements: concrete instances of the abstract classes and relations between them
- Query predicates:
  - `rdf(S, P, O)` returns only exactly matching triples
  - `rdf_has(S, P, O)` also takes the `subPropertyOf` relation into account, returning matches for all specializations of `P`
  - `rdf_reachable(S, P, O)` further considers transitivity of `P`
  - `rdf_triple(P, S, O)` additionally includes computables
  - `owl_has(S, P, O)` includes OWL inference results (e.g. inferred class membership of an individual)

Important: Most predicates can be used with different combinations of free and bound variables:

- `owl_individual_of(Ind, knowrob:'Cup')` reads all instances of class Cup
- `owl_individual_of(knowrob:'cup0', Class)` reads all classes of `cup0`
Example queries: ABOX

?- owl_has(A, rdf:type, knowrob:'Drawer').
A = 'http://ias.cs.tum.edu/kb/knowrob.owl#Drawer1' ;
A = 'http://ias.cs.tum.edu/kb/knowrob.owl#Drawer103' ;

?- owl_has(knowrob:'Drawer1', P, O).
P = 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
O = 'http://ias.cs.tum.edu/kb/knowrob.owl#Drawer' ;
P = 'http://ias.cs.tum.edu/kb/knowrob.owl#widthOfObject',
O = literal(type(xsd:float, '0.58045006')) ;
P = 'http://ias.cs.tum.edu/kb/knowrob.owl#properPhysicalParts',
O = 'http://ias.cs.tum.edu/kb/knowrob.owl#Door4' ;
Querying KnowRob from your program

- Tell/ask interface via a ROS service
  - **tell**: add statements
  - **ask**: send a query

- Send queries as string like you would do from a console

- Result serialized using json

- Client libraries for C++, Python, Java, Lisp available

---

**Example json_prolog launch file**

```xml
<launch>
  <node name="knowrob" pkg="rosprolog" type="run_with_prolog_env"
  args="demo_cotesys_fall2010 $(find json_prolog)/bin/json_prolog" />
</launch>
```

(see also [http://ros.org/wiki/json_prolog](http://ros.org/wiki/json_prolog))
# Querying KnowRob from Python

```python
#/usr/bin/env python

import roslib; roslib.load_manifest('json_prolog')
import rospy
import json_prolog

if __name__ == '__main__':
    rospy.init_node('test_json_prolog')
    prolog = json_prolog.Prolog()

    query = prolog.query("member(A, [1, 2, 3, 4]), B = ['x', A]"")

    for sol in query.solutions():
        print 'Found solution. A = %s, B = %s' % (sol['A'], sol['B'])

    query.finish()
```
Querying KnowRob from C++

```cpp
#include <string>
#include <iostream>
#include <ros/ros.h>
#include <json_prolog/prolog.h>

using namespace std;
using namespace json_prolog;

int main(int argc, char *argv[]) {
    ros::init(argc, argv, "test_json_prolog");

    Prolog pl;

    PrologQueryProxy bdgs = pl.query("member(A, [1, 2, 3]), B = ['x', A]");

    for(PrologQueryProxy::iterator it=bdgs.begin(); it != bdgs.end(); it++) {
        PrologBindings bdg = *it;
        cout << "Found solution: " << (bool)(it == bdgs.end()) << endl;
        cout << "A = " << bdg["A"] << endl;
        cout << "B = " << bdg["B"] << endl;
    }
    return 0;
}
```
Workspace setup and debugging

- Global history: put the following .plrc file in your home

```prolog
.plrc

rl_write_history :-
    expand_file_name("~/pl-history", [File|_]),
    rl_write_history(File).
:- ( current_prolog_flag(readline, true) ->
    expand_file_name("~/pl-history", [File|_]),
    (exists_file(File) -> rl_read_history(File); true),
    at_halt(rl_write_history); true ).
```

- Debugging: graphical tracer available in SWI Prolog

```prolog
Graphical tracer

?- guitracer.
?- trace.

(see also http://www.swi-prolog.org/pldoc/refman, Section 3.5)
```
Exercises (45min):

KnowRob ontology walkthrough: Overview of main classes and properties in http://ias.in.tum.de/kb/knowrob.owl

Structure of a KnowRob ROS package

KnowRob basics tutorial:
http://ias.in.tum.de/kb/wiki/index.php/Tutorial:_KnowRob_basics

Loading OWL files and ROS modules:
http://ias.in.tum.de/kb/wiki/index.php/Loading_files_and_ros_packages

Compile and run a json_prolog example client in your favorite language (launch the KnowRob service with roslaunch knowrob_tutorial tutorial.launch)
Further links

API documentation of KnowRob packages:
http://ias.cs.tum.edu/kb/api/\(<\text{pkg-name}\>/\)

Integration of common DL reasoners (e.g. for inferring class subsumption)
http://ias.in.tum.de/kb/wiki/index.php/OWL_reasoners_via_OWLAPI
http://owlapi.sourceforge.net/documentation.html

PLDOC source code documentation:
http://ias.in.tum.de/kb/wiki/index.php/Source_code_documentation_with_PLDOC

Prolog documentation (incl. semweb library and thea OWL library)
http://www.swi-prolog.org/pldoc/refman/
http://www.semanticweb.gr/thea/index.html

Matlab interface (possibly outdated)
http://ias.in.tum.de/kb/wiki/index.php/Using_MATLAB_from_Prolog
Outline: KnowRob tutorial

- **Introduction**: What can CRAM, KnowRob, etc be used for?
- **Languages**: Prolog, OWL, Java, etc
- **Basics**: launching KnowRob, loading files, querying, debugging
- **Objects**: class taxonomy, instances, components, environment models, units and coordinates, qualitative spatial relations, computable properties
- **Actions**: class-level descriptions, properties, projection, planning, action dependencies
- **RoboEarth**: concept, interface to KnowRob, up- and download of information, capability matching
- **ProbCog**: interface to statistical relation inference (Dominik)
Object representation

Objectives:

- Class-level knowledge about object types
- Instance-level representations of object configurations
- Metric as well as qualitative spatial information (+ conversion)
- Explicit representation of units and coordinates (+ conversion)
- Nominal object locations in an environment
- Changes of locations over time

(see also http://ias.in.tum.de/kb/wiki/index.php/Reasoning_about_objects)
Visualizing objects

- A java-based visualization canvas can display objects and their locations
- `visualization_canvas(C)` starts the canvas and stores a handle in the variable $C$
- `$c` refers to the last binding of the variable $C$
- `add_object(Identifier, $C)` sends an object instance to the canvas in order to visualize it

(see also API documentation: http://ias.cs.tum.edu/kb/api/mod_vis/)
Example queries: Visualization

?- owl_has(A, rdf:type, knowrob:'Drawer'),
   add_object_with_children(A, $C).

A = 'http://ias.cs.tum.edu/kb/knowrob.owl#Drawer1' ;
A = 'http://ias.cs.tum.edu/kb/knowrob.owl#Drawer103' ;
Semantic map representation

Abstract knowledge about object classes

Object instances and component hierarchy

Poses in the environment and their changes over time

Related: TBOX/SBOX, Galindo et al (RAS 2008)
Semantic map representation

Abstract knowledge about object classes

Object instances and component hierarchy

Poses in the environment and their changes over time
Spatio-temporal object representation

(see also http://ias.in.tum.de/kb/wiki/index.php/Modeling_perceived_or_inferred_objects)
Integrating perception

(compute-based interface)

- tabletop_obj_detector
- compTabletopObj
  service: /tabletop_obj_rec
- VisualPerception-42
  objectActedOn: icetea2
  eventOccursAt: homography9
  startTime: timep_1.572

(KNOWLEDGE BASE)

- homography9
  1 0 0 2.56
  0 1 0 1.32
  0 0 1 0.93
  0 0 0 1

- homography8
  1 0 0 2.34
  0 1 0 1.75
  0 0 1 0.73
  0 0 0 1

- homography5
  1 0 0 0.94
  0 1 0 1.05
  0 0 1 1.53
  0 0 0 1

- homography3
  1 0 0 3.04
  0 1 0 2.25
  0 0 1 0.93
  0 0 0 1

(see also)

http://ias.in.tum.de/kb/wiki/index.php/Modeling_perceived_or_inferred_objects)
Semantic map representation

Abstract knowledge about object classes

Object instances and component hierarchy

Poses in the environment and their changes over time
Representing articulated objects

- Poses and joint limits of hinges and prismatic joints
- Enable robot to open doors and drawers
Object part composition

- Part-of hierarchy: composition of objects from parts
- Hinged-to/fixed-to: kinematic properties
Integrating CAD models

- Visualization and computation of qualitative spatial relations
- Soon: segmentation and interpretation of object parts

(see also http://ias.in.tum.de/kb/wiki/index.php/CAD_Models)
Computing qualitative spatial relations

(see also http://ias.in.tum.de/KB/wiki/Inference/Spatial:_Computables)
Computing qualitative spatial relations

(see also http://ias.in.tum.de/KB/wiki/index.php/Tutorial:_Computables)
Computables

- Procedural attachment to OWL classes and properties
  - Computable properties:
    - Calculate if relation exists between two instances
    - Generate instances for which relation exists
  - Computable classes:
    - Check for class subsumption
    - Generate individuals of a class
Defining computables

- Define OWL property (as usual incl. Domain and Range)

  \[
  \textbf{ObjectProperty: after} \\
  \quad \textbf{SubPropertyOf: temporallyRelated} \\
  \quad \textbf{Domain: TimePoint} \\
  \quad \textbf{Range: TimePoint}
  \]

- Write Prolog predicate that computes this relation

  \[
  \text{comp_after(Pre, After) :-} \\
  \quad \text{owl_has(Pre, type, 'TimePoint'),} \\
  \quad \text{owl_has(After, type, 'TimePoint'),} \\
  \quad \text{term_to_atom(P, Pre),} \\
  \quad \text{term_to_atom(A, After),} \\
  \quad P < A.
  \]

- Attach this predicate to the OWL property

  \[
  \textbf{Individual: computeAfter} \\
  \quad \textbf{Types: PrologProperty} \\
  \quad \textbf{Facts: target after} \\
  \quad \text{command comp_after} \\
  \]

(see also http://ias.in.tum.de/kb/wiki/index.php/Define_computables)
Calling computables

- Computables can trigger complex calculations, include them only if needed
- Special predicate `rdf_triple(?P,?S,?O)` that returns the union of
  - asserted knowledge (via `owl_has(?S,?P,?O)`)  
  - results of computables (`rdfs_computable_triple(?S,?P,?O)`)  
  - provides a hook for custom predicates.

\[
\text{rdf_triple}(\text{Prop}, \text{Subj}, \text{Obj}) : - \\
\text{subproperty_of}(\text{SubProp}, \text{Prop}), \\
(\text{owl_has} (\text{Subj}, \text{SubProp}, \text{Obj}) ; \\
\text{rdfs_computable_triple}(\text{Subj}, \text{SubProp}, \text{Obj}) ; \\
\text{rdf_triple_hook}(\text{Subj}, \text{SubProp}, \text{Obj}) ).
\]

(see also http://ias.in.tum.de/kb/wiki/index.php/Tutorial:_Computables)
Semantic map representation

Abstract knowledge about object classes

Object instances and component hierarchy

Poses in the environment and their changes over time
Video: Unpacking a shopping basket

Thanks to Dejan Pangercic, Máthé Koppány, Zoltan-Csaba Marton, Lucian Goron, Monica Opris and Thomas Rühr for making this live demonstration possible.
Infer storage location based on generic class knowledge
Object ontology

- Automatically created ontology of >7500 objects from the online shop germandeli.com
- Class hierarchy from categories + perishability, weight, price, origin, ...
- SIFT recognition models from product pictures (work by Dejan Pangeric)

(see also http://www.ros.org/wiki/comp_germandeli)
Infer storage locations based on semantic object similarity

Learning Organizational Principles in Human Environments.
Martin Schuster, Dominik Jain, Moritz Tenorth and Michael Beetz. ICRA 2012

(see also http://www.ros.org/wiki/comp_orgprinciples)
Infer storage locations based on semantic object similarity

?- highlight_best_location_dtree(
orgprinciples:’CoffeeFilter1’, Canvas).

Best location: knowrob:Drawer7
Objects at location knowrob:Drawer7:
WUP similarity: object (class)
0.87500: orgprinciples:CoffeeGround1
(germandeli:Dallmayr_Classic_Ground_Coffee_250g)
0.75000: orgprinciples:EspressoBeans1
(germandeli:illy_Espresso_Whole_Beans_88_oz)
0.70588: orgprinciples:Sugar1
(germandeli:Nordzucker_Brauner_Teezucker_500g)
0.66667: orgprinciples:Tea2
(germandeli:Teekanne_Rotbusch_Tee_Vanille_20_Bags)
Open the drawer where cups are stored

- Infer most likely storage location
- Read articulation model from semantic map
Check if objects are placed correctly

- Infer most likely storage location
- Compare with actual locations of objects
Select objects based on their purpose

- Combine abstract object knowledge with map information
- Here: “appliance that can be used for washing dishes”
Integrate with human tracking data

- Observations of humans are represented in the KB
- Reason about their interaction with objects in the map
Scaling up to large environments

Searching Objects in Large-Scale Indoor Environments: A Decision-Theoretic Approach.
Lars Kunze, Michael Beetz, Manabu Saito, Haseru Azuma, Kei Okada, Masayuki Inaba. ICRA 2012
Exercises (45min):

Walkthrough: object pose representation

KnowRob objects tutorial:
http://ias.in.tum.de/kb/wiki/index.php/Reasoning_about_objects

Additional information:
http://ias.in.tum.de/kb/wiki/index.php/Measurement_units
http://ias.in.tum.de/kb/wiki/index.php/Coordinate_systems
http://ias.in.tum.de/kb/wiki/index.php/Map_data_representation
http://ias.in.tum.de/kb/wiki/index.php/CAD_Models
http://ias.in.tum.de/kb/wiki/index.php/Mod_semantic_map
Outline: KnowRob tutorial

- **Introduction:** What can CRAM, KnowRob, etc be used for?
- **Languages:** Prolog, OWL, Java, etc
- **Basics:** launching KnowRob, loading files, querying, debugging
- **Objects:** class taxonomy, instances, components, environment models, units and coordinates, qualitative spatial relations, computable properties
- **Actions:** class-level descriptions, properties, projection, planning, action dependencies
- **RoboEarth:** concept, interface to KnowRob, up- and download of information, capability matching
- **ProbCog:** interface to statistical relation inference (Dominik)
Action representation

Objectives:

- Different hierarchies: temporal composition, class taxonomy
- Action properties: toLocation, objectActedOn, bodyPartUsed,...
- Partial ordering of actions in a task
- Reason about effects of actions on objects
- Support projection and (simple) planning
- Generate executable robot plan from specification
- Specify requirements on robot capabilities or components

(see also http://ias.in.tum.de/kb/wiki/index.php/Reasoning_about_actions)
Reasoning about action sequences

Generate initial task description from Web instructions
Reasoning about action sequences

- Project action effects to infer interaction with objects
- Specific properties exactly describe object-action relation
- Reason about changes induced on objects
Reasoning about action sequences

- Complete action sequences using background knowledge
- Infer storage location, opening trajectory, etc
Reasoning about action sequences

- Detect knowledge gaps by comparing projected world state with required action inputs
- Apply planning to add required actions (see also knowrob_actions package)
Reasoning about action sequences

- Include physics-based reasoning to infer action parameterizations
- Work by Lars Kunze, Mihai Dolha
Action class representation

- Generic action class definition, incl. super-classes and sub-actions

  Class: PuttingSomethingSomewhere
  SubClassOf:
    Movement - TranslationEvent
    TransportationEvent
  subAction some PickingUpAnObject
  subAction some CarryingWhileLocomoting
  subAction some PuttingDownAnObject
  orderingConstraints value SubEventOrdering1
  orderingConstraints value SubEventOrdering2

- Specification of sub-actions and ordering constraints

  Individual: SubEventOrdering1
  Types:
    PartialOrdering - Strict
  Facts:
    occursBeforeInOrdering PickingUpAnObject
    occursAfterInOrdering CarryingWhileLocomoting
Composing actions to tasks

**Class**: SetATable
**Annotations**: label "set a table"
**SubClassOf**: Action
**EquivalentTo**:
- subAction some PutPlaceMatInFrontOfChair
- subAction some PutPlateInCenterOfPlaceMat
- subAction some PutKnifeRightOfPlate
- subAction some [...] orderingConstraints value [...] 

**Class**: PutPlaceMatInFrontOfChair
**EquivalentTo**:
- PuttingSomethingSomewhere
  - objectActedOn value PlaceMat1
toLocation some Place1

**Class**: Place1
**EquivalentTo**:
inFrontOf - Generally some Chair - PieceOfFurniture

**Individual**: PlaceMat1
**Types**: PlaceMat
Task specification

ServeADrink
dependsOnComponent ObjectRecognitionModel AND providesModelFor.Bottle
dependsOnComponent ObjectRecognitionModel AND providesModelFor.Bed
dependsOnComponent ObjectRecognitionModel AND providesModelFor.Cabinet

GraspBottle
subClassOf GraspingSomething
objectActedOn bottle1

ReachToApproachPose
subClassOf toLocation
Reaching handPose-approachObject

OpenGripperForGrasping
subClassOf toLocation
OpeningAGripper handPose-approachObject

ReachForObjectPose
subClassOf toLocation
Reaching handPose-graspObject

CloseGripperForGrasping
subClassOf toLocation
ClosingAGripper handPose-approachObject

LiftObjectToApproachPose
subClassOf toLocation
Reaching handPose-approachObject

reduce
CRAM plan generation

```
(def-top-level-plan serve-a-drink ()
(with-designators (
  (bottle1 object '((name bottle1)
                   (type drinking-bottle)))))
  (bed1 object '((name bed1)
                (type bed-piece-of-furniture)))))
  (hand-pose-handover1 (location ‘((on , bed1))))
  (robot-pose-handover1 (location ‘((to reach)
                                (side : right)
                                (loc , hand-pose-handover1))))
  (arms-at101 (action ‘((type trajectory)
                        (pose , hand-pose-handover1)
                        (side : right)))))
  (unhand-action102 (action ‘((type open-gripper)
                           (side : right)))))
)

(achieve ‘(object-in-hand , bottle1 : right))
(at-location (robot-pose-handover1))
(achieve ‘(arms-at , arms-at101))
(achieve ‘(arms-at , unhand-action102))))
```
Requirements of actions on robot capabilities

Objectives:

- Find out if robot can do a job or, if not, what is missing
- Technically: flexible matching between robot capabilities and action requirements

Complex problem because of:

- hierarchical structure of robot components
- temporal action hierarchy (task composed of sub-actions)
- inheritance hierarchy (requirements defined on higher levels)
- inter-dependencies (capabilities depend on other components or capabilities)
- tasks do not usually come with requirement specs, they need to be inferred using prior knowledge
SRDL: Robot description

- Kinematic robot model imported from URDF
  - Links and joints, incl. coordinate transformations
  - linked by sub-properties of transitive subComponent property
- Manual semantic annotations: 'these links form a gripper'
- Robot capabilities, e.g. 2D navigation
  (see also http://ias.in.tum.de/kb/wiki/index.php/SRDL2_Tutorial)
SRDL: Query for available robot components

\[ \text{sub\_component}(\text{Super}, \text{Sub}) : \neg \text{owl\_has}(\text{Super}, \text{srdl2comp: 'subComponent'}, \text{Sub}). \]

\[ \text{sub\_component}(\text{Super}, \text{Sub}) : \neg \text{owl\_has}(\text{Sub}, \text{srdl2comp: 'baseLinkOfComposition'}, \text{Base}), \]
\[ \quad \text{sub\_component}(\text{Super}, \text{Base}), \]
\[ \quad \text{owl\_has}(\text{Sub}, \text{srdl2comp: 'endLinkOfComposition'}, \text{End}), \]
\[ \quad \text{sub\_component}(\text{Super}, \text{End}). \]

\[ \text{comp\_type\_available}(\text{Super}, \text{SubT}) : \neg \]
\[ \quad \text{sub\_component}(\text{Super}, \text{Sub}), \]
\[ \quad \text{owl\_individual\_of}(\text{Sub}, \text{SubT}). \]

- Very simple queries for reading components and checking availability
- Query exploits transitivity of subComponent property
SRDL: Query for available capabilities

% capability asserted for robot instance
\texttt{cap\_available\_on\_robot(Cap,Robot) :-}
\texttt{owl\_has(Robot, srdl2cap: 'hasCapability', SubCap),}
\texttt{owl\_subclass\_of(SubCap, Cap).}

% capability asserted for robot class
\texttt{cap\_available\_on\_robot(Cap,Robot) :-}
\texttt{rdfs\_individual\_of(Robot, RobotClass),}
\texttt{class\_properties(RobotClass, srdl2cap: 'hasCapability', SubCap),}
\texttt{owl\_subclass\_of(SubCap, Cap).}

% capability depends only on available components or capabilities
\texttt{cap\_available\_on\_robot(Cap,Robot) :-}
\texttt{rdfs\_subclass\_of(Cap, srdl2cap: 'Capability'),}
\texttt{forall(class\_properties(Cap, srdl2comp: 'dependsOnComponent', CompT),}
\texttt{comp\_type\_available(Robot, CompT)),}
\texttt{forall(class\_properties(Cap, srdl2cap: 'dependsOnCapability', SubCap),}
\texttt{cap\_available\_on\_robot(SubCap, Robot)).}
SRDL: Query for required components

- Next step: read required components from action specification
- Recursively read dependencies for sub-actions
- Inherit dependencies from super-classes
- Equivalent for capability dependencies

% directly required by an action and its sub-actions
required_comp_for_action(Action, Comp) :-
class_properties(Action, srdl2comp: 'dependsOnComponent', Comp).

% required by any of its sub-actions
required_comp_for_action(Action, Comp) :-
plan_subevents_recursive(Action, SubAction),
class_properties(SubAction, srdl2comp: 'dependsOnComponent', Comp).

% indirectly required by required capabilities
required_comp_for_action(Action, Comp) :-
required_cap_for_action(Action, Cap),
class_properties(Cap, srdl2comp: 'dependsOnComponent', Comp).
SRDL: Determine missing components

- Next: determine missing components (required but not available)
- Availability is recursively evaluated: available capabilities are defined as 'depend only on available ones'
- Empty result: nothing missing \(\rightarrow\) action feasible

```prolog
missing_comp_for_action(Action, Robot, Comp) :-
  required_comp_for_action(Action, Comp),
  \+ comp_type_available(Robot, Comp).

missing_cap_for_action(Action, Robot, Cap) :-
  required_cap_for_action(Action, Cap),
  \+ cap_available_on_robot(Cap, Robot).

action_feasible_on_robot(Action, Robot) :-
  \+ missing_cap_for_action(Action, Robot, _),
  \+ missing_comp_for_action(Action, Robot, _).
```
SRDL: Lessons learned

- Inference is supported by modeling:
  - Transitivity: read whole branches of kinematic tree at once
  - Property hierarchy: describe specifically, query generically
  - Class taxonomy: simply inherit dependencies from superclasses

- Task specifications do not need any special annotation
  - Dependencies inherited from generic action classes
  - Action dependencies are stored locally and are thus reusable

- Prolog facilitates definition of additional solutions:
  - E.g. determine available capabilities by looking at the ROS network

- Easy implementation of complex behavior
  - Few simple Prolog rules and a bit of OWL modeling
  - Robot model largely imported automatically
Exercises (30min):

KnowRob actions tutorial:
http://ias.in.tum.de/kb/wiki/index.php/Reasoning_about_actions

Additional information:
http://ias.in.tum.de/kb/wiki/index.php/Modeling_tasks_and_actions
http://ias.in.tum.de/kb/wiki/index.php/Modeling_changing_objects
Outline: KnowRob tutorial

- **Introduction:** What can CRAM, KnowRob, etc be used for?
- **Languages:** Prolog, OWL, Java, etc
- **Basics:** launching KnowRob, loading files, querying, debugging
- **Objects:** class taxonomy, instances, components, environment models, units and coordinates, qualitative spatial relations, computable properties
- **Actions:** class-level descriptions, properties, projection, planning, action dependencies
- **RoboEarth:** concept, interface to KnowRob, up- and download of information, capability matching
- **ProbCog:** interface to statistical relation inference (Dominik)
Information exchange among humans

Informal task instructions in natural language

Execution requires common-sense knowledge
Information exchange among robots

What to export?
Information exchange among robots

What to export?

How to represent information?
Information exchange among robots

What to export?

How to represent information?

How to find information?

RoboEarth
Information exchange among robots

What to export?

How to find information?

How to represent information?

How to execute?
RoboEarth

Action recipes

Object models

Environment maps

Web-service client interface

Semantic Robot Description Language

Spatio-temporal Environment Models

Action recipes

Translation between recipes and CPL plans

Query interface for knowledge-enabled task execution

Plan-based robot controller

KnowRob

(see also

http://ias.in.tum.de/kb/wiki/index.php/Exchanging_information_via_RoboEarth)
Application of SRDL in RoboEarth
Video: Downloading recipes, objects, and maps
Object models

Initialization phase
- Recognition model (SURF, point cloud, ...)
- OWL representation (semantic properties, components)
- CAD model (visualization and reasoning)

TBOX object models

KnowRob object management module

Detection phase

Estimation phase
- re_articulation
- update joint information (pose, joint limits, ...)

Upload phase
- extract TBOX object model
Finding maps for an environment
Exercises (45min):

KnowRob RoboEarth tutorial:
http://ias.in.tum.de/kb/wiki/index.php/Exchanging_information_via_RoboEarth

Additional information:
http://www.ros.org/wiki/roboearth
http://www.ros.org/wiki/re_comm
Outline: KnowRob tutorial

- **Introduction**: What can CRAM, KnowRob, etc be used for?
- **Languages**: Prolog, OWL, Java, etc
- **Basics**: launching KnowRob, loading files, querying, debugging
- **Objects**: class taxonomy, instances, components, environment models, units and coordinates, qualitative spatial relations, computable properties
- **Actions**: class-level descriptions, properties, projection, planning, action dependencies
- **RoboEarth**: concept, interface to KnowRob, up- and download of information, capability matching
- **ProbCog**: interface to statistical relation inference (Dominik)
Interface to statistical relational learning and inference

Objectives:

- Integrate probabilistic logical models to account for uncertain information
- Query for results generated by statistical relational models
- Provide evidence for statistical inference to ProbCog

(see also http://ias.in.tum.de/research/probcog)
ProbCog interface

KnowRob - ProbCog interface

performInference
- read evidence using the predicates in the prolog module
  * tablesetting:usesAnyIn(…)
- load the ProbCog model, set the evidence, set open/closed world predicates, run inference
- map the results to KnowRob names using the probcog2knowrob(S,K) predicate

ProbCog server
- tablesetting
- kitchenlayout
- partialorder

Prolog probcog module
- determine suitable model
  * either based on query param
  * or default model for this predicate
  (defined in the sridb module as OWL or Prolog statements)
- load corresponding prolog module

KnowRob - ProbCog interface

tablesetting
- wrappers for advanced queries
- wrappers for reading evidence:
  usesAnyIn(…):-
  rdf_has(A,..B),
  ...
- probcog2knowrob

KNOWROB

kitchenlayout

partialorder
Thank you for your attention