Knowledge Processing for Autonomous Robots

Moritz Tenorth
Intelligent Autonomous Systems Group
Technische Universität München

November 5, 2010
1. Why Knowledge Processing for Autonomous Robots?

2. Introduction to Prolog

3. Overview of KnowRob

4. Asking Queries

5. Extending KnowRob

6. Integrating Perception Components

7. Inferring Control Decisions with KnowRob

8. Outlook
Outline

1. Why Knowledge Processing for Autonomous Robots?
2. Introduction to Prolog
3. Overview of KnowRob
4. Asking Queries
5. Extending KnowRob
6. Integrating Perception Components
7. Inferring Control Decisions with KnowRob
8. Outlook
Knowing what you are looking at

- Can you eat it or drink it?
- Is it perishable?
- Is it fragile?
- Can you recognize it?
- Where would you put it?
- How much is it?
- Have you seen anything similar before?
Knowing what you are looking at

Not only labels, but

- **Sub-class relations**: electrical appliance, household item, cooling device
- **Properties of objects**: fragile, heavy, perishable, edible
- **Relations to other objects**: storage location, similarity
- **Integration of knowledge sources**: common sense, environment, perception, human dialog
Knowing where to look for something

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

- **Basic:**
  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

- **Basic:**
  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
1. Why Knowledge Processing for Autonomous Robots?

2. Introduction to Prolog

3. Overview of KnowRob

4. Asking Queries

5. Extending KnowRob

6. Integrating Perception Components

7. Inferring Control Decisions with KnowRob

8. Outlook
Why Prolog for autonomous robots?

- Can be read declaratively
- Can be programmed (procedurally)
- There are textbooks and tutorials
- Industrial-strength open-source implementations
- Foreign language interfaces
- Result interpretation: similar to a DB query language
Knowledge base for the milk example

milk.pl

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).
Two main types of queries:
- Query with unbound variables $\rightarrow$ variable bindings
- All variables bound $\rightarrow$ true/false result

Example queries

```
$ 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).
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

?- 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

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).
Predicte 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

dairyproduct(milk).
perishable(Stuff) :- storedIn(Stuff, Ref), refrigerator(Ref).
1. Why Knowledge Processing for Autonomous Robots?

2. Introduction to Prolog

3. Overview of KnowRob

4. Asking Queries

5. Extending KnowRob

6. Integrating Perception Components

7. Inferring Control Decisions with KnowRob

8. Outlook
The KnowRob knowledge base

- **Query interface**
- **Visualization modules**
- **Common-sense knowledge**
- **Encyclopedic knowledge**

**KNOW-ROB**

- Action abstraction
- Action-related concepts
- Physical reasoning

**Reasoners**
- DL, MLN, BLN

**Machine learning**
- Clustering,
- Classification,
- Segmentation

**Natural language processing**

**Computable classes and properties**
- ROS
- SQL

November 5, 2010

Knowledge Processing for Autonomous Robots

Moritz Tenorth
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:
  ```
  rosrn `rosprolog rosprolog knowrob_tutorial`
  ```
Outline

1. Why Knowledge Processing for Autonomous Robots?

2. Introduction to Prolog

3. Overview of KnowRob

4. Asking Queries

5. Extending KnowRob

6. Integrating Perception Components

7. Inferring Control Decisions with KnowRob

8. Outlook

November 5, 2010
Knowledge Processing for Autonomous Robots
Moritz Tenorth
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
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()
```

November 5, 2010

Knowledge Processing for Autonomous Robots

Moritz Tenorth
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;
}
```
The KnowRob Taxonomy

- Defines the basic concepts of the household and robotic domain
- Hierarchy of classes and their properties
- Stored in the Web Ontology Language OWL as (S,P,O) triples
The KnowRob Taxonomy

- kitchen:ElectricalHouseholdAppliance
- kitchen:RefrigeratedStorageDevice
- kitchen:typePrimaryFunction-deviceUsed some kitchen:CoolingProcess
- kitchen:typePrimaryFunction-StoragePlaceFor some kitchen:FoodOrDrink
- kitchen:contains some kitchen:PartiallyTangible
- kitchen:containsCavity exactly 1 kitchen:Cavity
- kitchen:containsPortals some kitchen:Portal
- kitchen:hasEnclosure exactly 1 kitchen:Enclosure
- kitchen:physicalDecompositions some (kitchen:Enclosure or kitchen:Cavity or kitchen:Portal or kitchen:PortalConnection)
- kitchen:properPhysicalPartTypes some kitchen:ControlDevice
- kitchen:spatiallyRelated some kitchen:SpatialThing

November 5, 2010

Knowledge Processing for Autonomous Robots

Moritz Tenorth
Scaling up
Class Restrictions

Asserted vs. inferred class based on restrictions
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
Example queries: TBOX

Example

```
$ 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'
```
The ABOX contains **assertional statements**: concrete instances of the abstract classes and relations between them.

- `owl_individual_of(Ind, Class)` is true if `Ind` is an instance of `Class`.
- `owl_has(S, P, O)` is a general query predicate for relations `P` between a subject `S` and an object `O`.

**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 `cup0` belongs to.
Example queries: ABOX

?- owl_has(A, rdf:type, knowrob:'Cupboard').

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' ;
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.
- **API documentation**: [http://ias.in.tum.de/kb/api/mod_vis.pl.html](http://ias.in.tum.de/kb/api/mod_vis.pl.html)
Example queries: Visualization

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

A = 'http://ias.cs.tum.edu/kb/ias_semantic_map.owl#closet0' ;
A = 'http://ias.cs.tum.edu/kb/ias_semantic_map.owl#cupboard0' ;
Exercise sheet – Querying the Knowledge Base
Optional queries

- Sub-classes of CookingUtensil
- Everything that has a handle
- What to use for heating food?
- What needs electricity?
- What needs to be put into the fridge? (hint: predicate storagePlaceFor)
1. Why Knowledge Processing for Autonomous Robots?
2. Introduction to Prolog
3. Overview of KnowRob
4. Asking Queries
5. Extending KnowRob
6. Integrating Perception Components
7. Inferring Control Decisions with KnowRob
8. Outlook
Editing Ontologies with Protege

Graphical editor for OWL ontologies

Download and tutorials available at http://protege.stanford.edu/
Exercise sheet – Extending the Knowledge Base
1. Why Knowledge Processing for Autonomous Robots?
2. Introduction to Prolog
3. Overview of KnowRob
4. Asking Queries
5. Extending KnowRob
6. Integrating Perception Components
7. Inferring Control Decisions with KnowRob
8. Outlook
Reasoning on robot data

- Robot knowledge processing = dynamic knowledge processing
- Knowledge base needs to make decisions based on the most current information

Use the robot’s data structures as extended knowledge base!

- Create instances on demand (e.g. recognized objects)
- Compute properties of and relations between them
Computables: Attach computational methods to semantic relations

Two components:
- Prolog predicate to compute the relation
- OWL file to link the predicate to the OWL relation
Calling computables

- Take computables into account only when needed

- Special query predicates: `rdf_triple`, `rdfs_instance_of`
  - return all normal solutions (`owl_has`, `owl_individual_of`)
  - search for all computables for the class/relation
  - return all results of all computables

Useful predicates

- `rdf_triple(+Pred, ?Subj, ?Obj)`
- `rdfs_instance_of(?Inst, ?Class)`

Note the different order of arguments for `rdf_triple`
Task 1: Retrieve object information

- Java client to interface with ROS:
  `src/edu/tum/cs/ias/knowrob/tutorial/ROSClient.java`

- Use JPL to call the object detection service from Prolog
  `/object_detection`

**Useful predicates**

- `jpl_new(+Class, +Arguments, -Value)`
- `jpl_call(+Class, +Method, +Args, -Result)`
- `jpl_call(+Object, +Method, +Args, -Result)`
- `jpl_get(+Object, +Field, -Value)`
- `jpl_array_to_list(+JArray, -PList)`

See also: [http://www.swi-prolog.org/packages/jpl/prolog_api/api.html](http://www.swi-prolog.org/packages/jpl/prolog_api/api.html)
Task 1: Retrieve object information

- Extract object information from the TabletopDetectionResult
- Pass these values to the predicates for creating the object instance, the perception instance, and for linking those

**Useful predicates**

```prolog
create_perception_instance(-Perception)
create_object_instance(+ObjTypes, +ID, -ObjInst)
set_object_perception(+ObjInst, +Perception)
set_perception_pose(+Perception, [+M00, +M01, +M02, +M03, +M10, +M11, +M12, +M13, +M20, +M21, +M22, +M23, +M30, +M31, +M32, +M33])
```
Task 1: Object representation

- VisualPerception-42
  - objectActedOn: icetea2
  - eventOccursAt: homography9
  - startTime: timep_1.572
- VisualPerception-41
  - objectActedOn: icetea2
  - eventOccursAt: homography8
  - startTime: timep_1.493
- VisualPerception-37
  - objectActedOn: icetea2
  - eventOccursAt: homography5
  - startTime: timep_1.295
- VisualPerception-34
  - objectActedOn: icetea2
  - eventOccursAt: homography3
  - startTime: timep_1.243

- icetea2
  - type: TetraPak
  - width: 0.1
  - depth: 0.1
  - height: 0.3

Motivation Prolog KnowRob Query Extend Integrate Use Outlook

November 5, 2010
Knowledge Processing for Autonomous Robots Moritz Tenorth
Task 2: Reason about the perceived objects

Have a look at the KnowRob ontology and formulate the following queries on the just perceived set of objects:

- Everything that can contain other things
- All food items
- Perishable things
- The positions \((x,y,z)\) of all drinks

**Useful predicates**

- `rdfs_instance_of(A, knowrob:`HumanScaleObject')`
- `owl_individual_of(Inst, Class)`
- `owl_subclass_of(Sub, Super)`
- `rdf_triple(knowrob:orientation, Obj, Or)`
- `rdf_triple(knowrob:m03, Or, X)`
Optional tasks

- Query for items that belong into the fridge
- Determine the pose of the handle of the storage place for all detected perishable items
- Visualize your results in mod_vis
- Write a program in Python/C++ that sends the above queries via ROS
- Have a look at owl/tabletop_obj.owl to see how the computable is realized
1. Why Knowledge Processing for Autonomous Robots?

2. Introduction to Prolog

3. Overview of KnowRob

4. Asking Queries

5. Extending KnowRob

6. Integrating Perception Components

7. Inferring Control Decisions with KnowRob

8. Outlook
Integration into the executive

Lorenz
1. Why Knowledge Processing for Autonomous Robots?
2. Introduction to Prolog
3. Overview of KnowRob
4. Asking Queries
5. Extending KnowRob
6. Integrating Perception Components
7. Inferring Control Decisions with KnowRob
8. Outlook
Reasoning on objects

**comp_cop**
Listen to topics and add the information to the knowledge base

**comp.spatial, comp.temporal**
Compute qualitative spatial and temporal relations (in, on, during)
### comp_ehow

Translate natural-language instructions into OWL representations and robot plans

![WikiHow screenshot](image1.png)

### comp_fipm, mod_ham

Segment and abstract human mocap data

![Segmentation and abstract human data](image2.png)

---

November 5, 2010

Knowledge Processing for Autonomous Robots

Moritz Tenorth
**srldb, mod_prob_cog**

Probabilistic inference using Markov Logic Networks and Bayesian Logic Networks

**ias_prolog_addons/classifiers.pl**

Weka and Mallet libraries for classifiers and clusterers
**mod_similarity**

Compute the semantic similarity between concepts in the ontology

**ias_naive_physics_sim**

Naive physics reasoning for computing relations in the knowledge base
Thank you for your attention

Questions?