Chair for Cognitive Systems
Department of Electrical and Computer Engineering
Technische Universität München

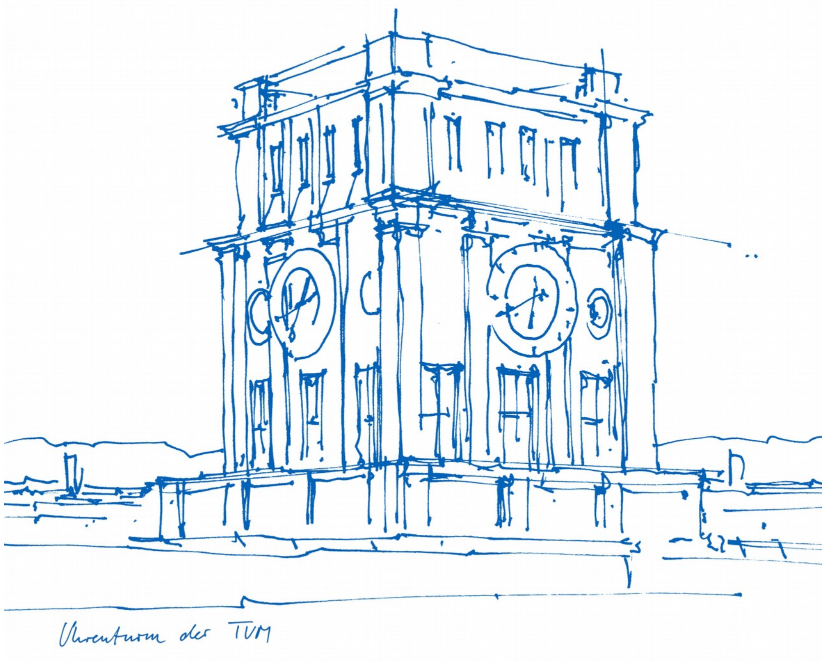
RoboCup@Home Practical course

Tutorials

Dr. Karinne Ramirez-Amaro
Dr. Emmanuel Dean
Dr. Pablo Lanillos
M.Sc. Roger Guadarrama
Dr. Gordon Cheng

Technical University of Munich

Chair for Cognitve Systems



Definition of teams

1) **Hands-on tutorials:** introductory tutorials to get familiar with the equipment (robot & sensors) used for this course.

2) **Group definition tasks:** form groups according to the students' knowledge to address different challenges of the competition. The students designate a team leader.

3) **Development and test phase:** design and implementation of algorithms to solve the problems defined for your working team.

4) **Final phase:** test real scenarios on a mobile robot to evaluate the performance of the robots abilities.

Definition of teams

2) Group definition tasks: form groups according to the students' knowledge to address different challenges of the competition.

- Designate a **team leader**.

Responsibilities of the team leader:

- Coordinate the work of the group. Make sure that the work is correctly distributed.
- Be responsible for the key of the laboratory
- Direct communication with the supervisor(s)

Definition of topics

Category III: This category includes:

- following a human,
- indoor navigation in crowded environments,
- recognizing & grasping alike objects,
- find a calling person (waving or shouting), etc.
- deal with incomplete information

IMPORTANT: Define the strongest capabilities of your team: Control, navigation, perception, and/or learning.

Tutorial 7: System integration

- **Object Recognition**
- **Robot Manipulation**
- **Mapping and Navigation**
- **Reasoning**

Tutorial 7: System integration- Reasoning

Exercise: The robot needs to move to the shelf grab one item and move it to the table.

Perception:

- 3D object localization
- Object detection
- Identify the shelf/table

Robot manipulation:

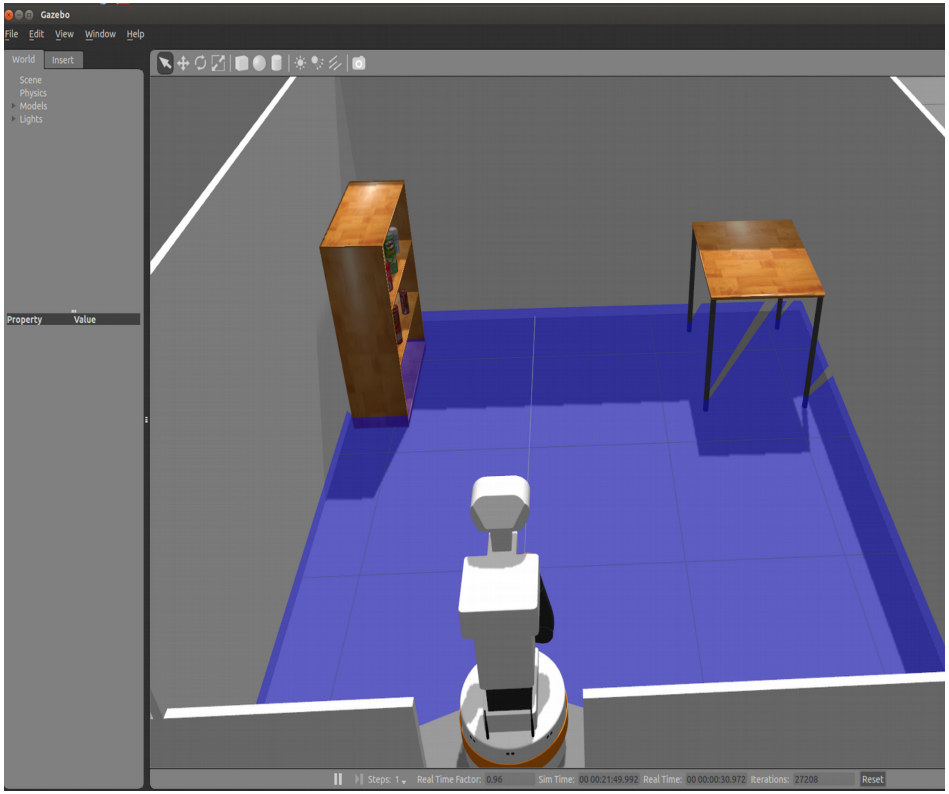
- Command the robot arm to a desired pose
- Grasp the object

Navigation:

- Move to the shelf and avoid collisions
- Move to the table

Reasoning:

- Remember the object properties
- Define the actions sequences

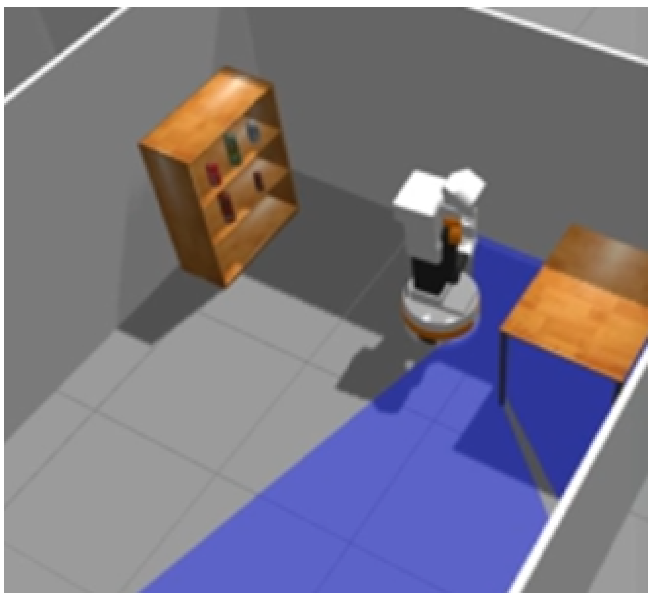


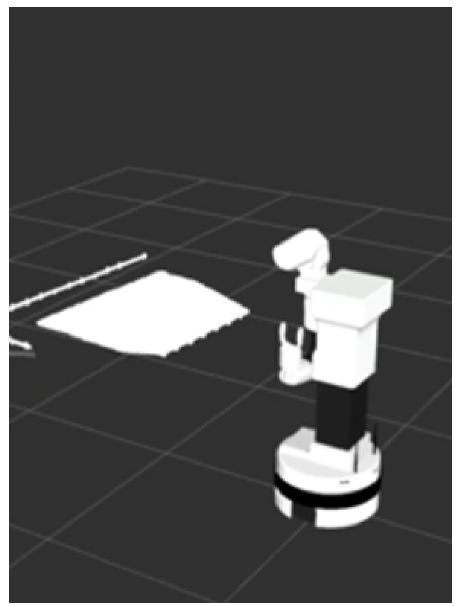
Tutorial 7: System integration

- **Object Recognition**

Perception

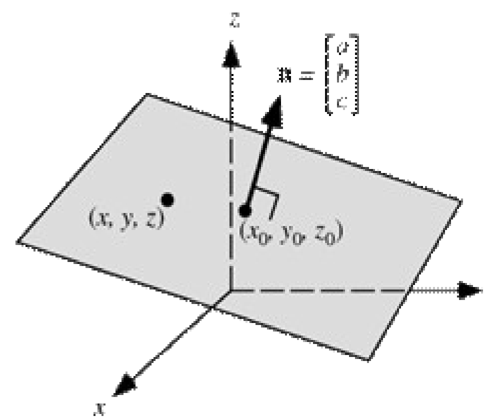
From plane segmentation to table computation





pcl::ModelCoefficients::Ptr

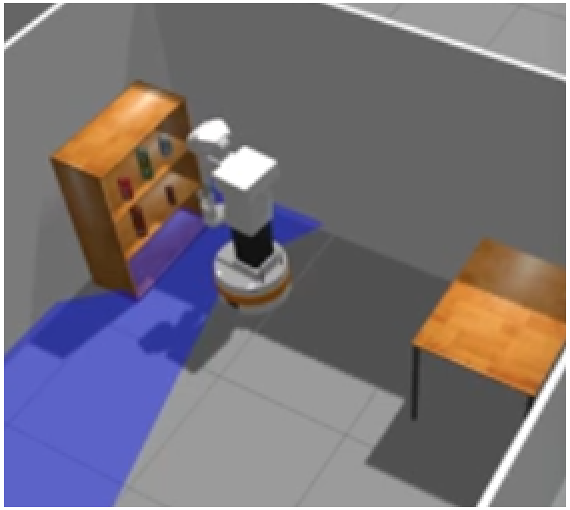
$$ax + by + cy + d = 0$$

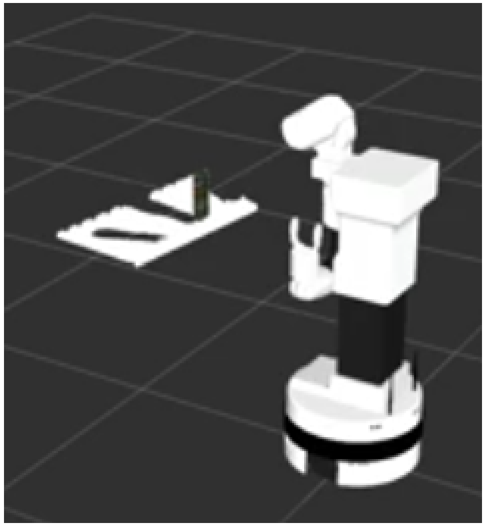


Chair for Cognitive Systems
Department of Electrical and Computer Engineering
Technische Universität München

Perception

Segmenting the objects on the bookshelf



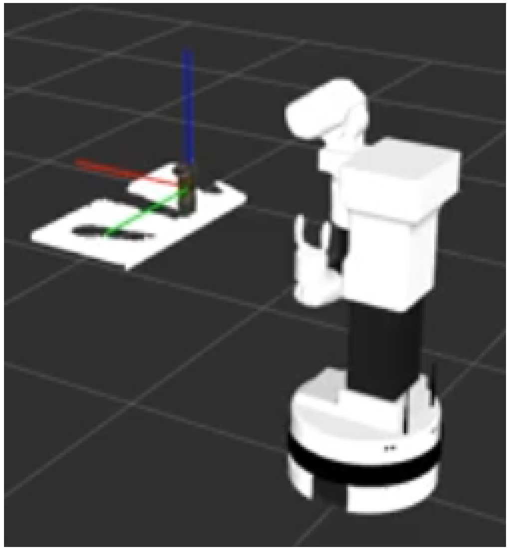


- Transform to frame if needed (pcl_ros::transformPointCloud)
- Filter limits (Passthrough)
- Downsample (VoxelGrid)
- Remove the planes (pcl::SACSegmentation)
- Find the cylinders (pcl::SACSegmentationFromNormals)
- Remove outliers (pcl::StatisticalOutlierRemoval)
- Publish the list of poinclouds or the one needed

Chair for Cognitive Systems
Department of Electrical and Computer Engineering
Technische Universität München

Perception

What else we can get from the Point Cloud data?



- Shape coefficients: cylinder [point_on_axis (x,y,z), axis_direction, (x,y,z) radius]
- 3D centroid
- Orientation
- Bounding box: min (x,y,z) and max (x,y,z) | centre (x0,y0,z0)
- Colored Texture: VoxelGrid.setDownsampleAllData(true);
- Descriptors
- Distance to the object

In order to grasp we need the **pose** of the object.

Chair for Cognitive Systems
Department of Electrical and Computer Engineering
Technische Universität München

Perception

Tips for visualization in rviz:

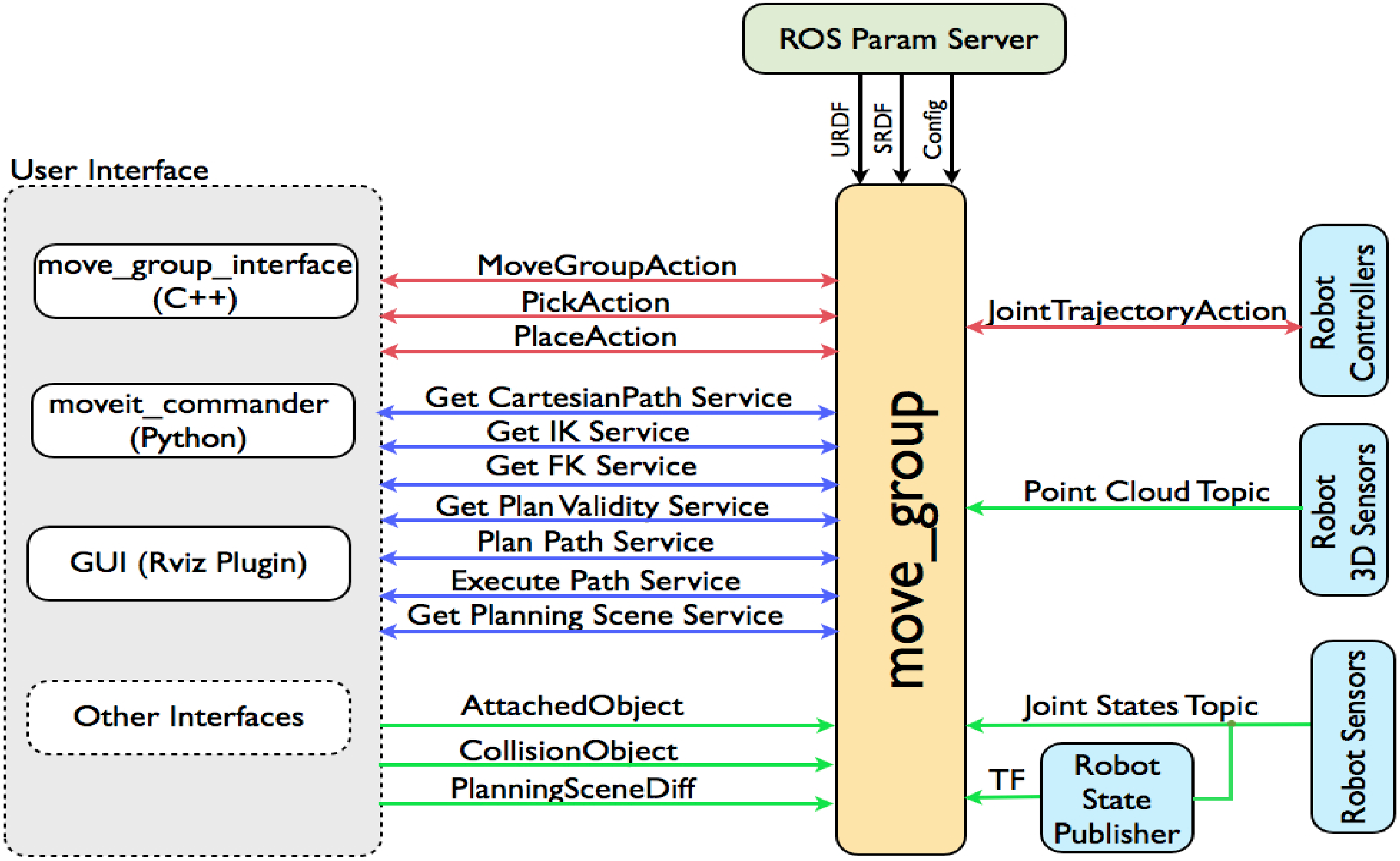
- One object: send the PointCloud2
- Several objects: use markers (visualization_msgs/MarkerArray.msg)

```
#ifdef PUBLISH_MARKERS
visualization_msgs::Marker ObjectClustering::getCloudMarker(const pcl::PointCloud<PointType>::Ptr  cloud, int id)
{
  //create the marker
  visualization_msgs::Marker marker;
  marker.header.frame_id = processing_frame_;
  marker.header.stamp = ros::Time();
  marker.action = visualization_msgs::Marker::ADD;
  marker.lifetime = ros::Duration(5);
  marker.ns = "segmentation";
  marker.id = id;
  marker.pose.orientation.w = 1;

  marker.type = visualization_msgs::Marker::POINTS;
  marker.scale.x = 0.002;
  marker.scale.y = 0.002;
  marker.scale.z = 1.0;

  marker.color.r = ((double)rand())/RAND_MAX;
  marker.color.g = ((double)rand())/RAND_MAX;
  marker.color.b = ((double)rand())/RAND_MAX;
  marker.color.a = 1.0;

  for(size_t i=0; i<cloud->size(); i++) {
    geometry_msgs::Point p;
    p.x = (*cloud)[i].x;
    p.y = (*cloud)[i].y;
    p.z = (*cloud)[i].z;
    marker.points.push_back(p);
  }

  return marker;
}
#endif
```

Tutorial 7: System integration

- **Object Recognition**
- **Robot Manipulation**

MoveIt! Motion planner



Chair for Cognitive Systems
Department of Electrical and Computer Engineering
Technische Universität München

Grasping with MoveIt! planner

1) Define object position (From perception node)
2) Feed the object position and geometry restrictions to the move_group node.
3) Request a planning service.
4) Wait for response.
5) Execute the motion.

6) Check MoveIt! Tutorial from http://wiki.ros.org/Robots/TIAGo/Tutorials

Tutorial 7: System integration

- **Object Recognition**
- **Robot Manipulation**
- **Mapping and Navigation**

Navigation and Mapping

Hints:

- Generate a map for the testing environment.
- Use the obtained map to do navigation.
- Use MoveBaseGoal to move the robot with respect to the base_link.

If you want to navigate in the unknown environment (without previously building a map) you can check another SLAM packages :

http://wiki.ros.org/hector_slam or http://wiki.ros.org/slam_karto

Tutorial 7: System integration

- Object Recognition
- Robot Manipulation
- Mapping and Navigation
- Reasoning

Tutorial 7: System integration- Reasoning

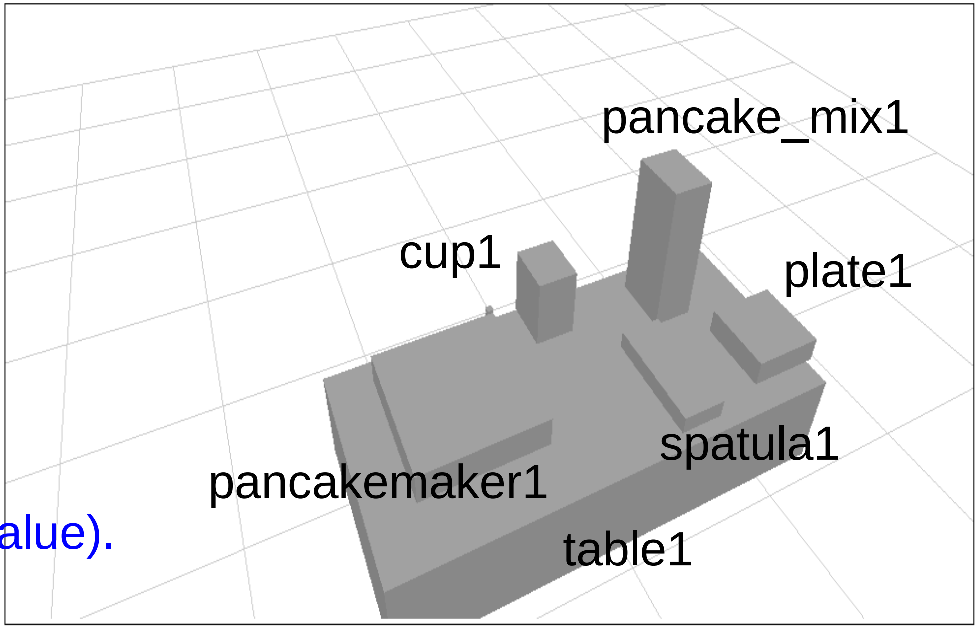
Imagine that you have the below scenario and you want to retrieve the objects on top of the table. Use the provided template to load a semantic map that contains the objects and positions for this scenario as shown in the figure.

Notes:

- The positions of the objects are located in the file: "semRoom_semantic_map.owl"
- To ask about the objects on top of the table use the query: comp_onTopTable(Obj,Value).
- To retrieve the object properties use the following query:

rdf_triple(knowrob:'aboveOfTable',Obj, Value).

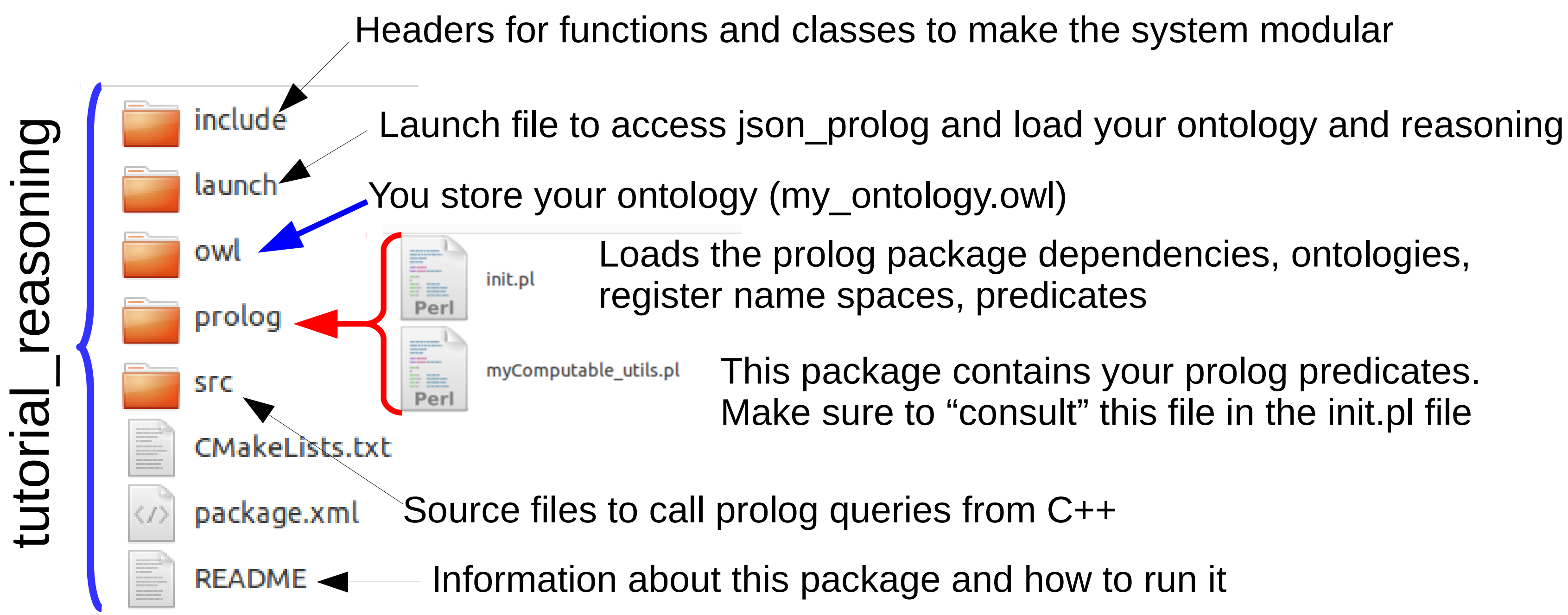
KnowRob Web Visualization



Chair for Cognitive Systems
Department of Electrical and Computer Engineering
Technische Universität München

Tutorial 7: System integration- Reasoning

Download and compile the provided template.



Tutorial 7: System integration- Reasoning

Remember that our reasoning system is programed in Prolog, then first we need to make sure that the Prolog statements are correct.

1) Verify that our package loads the ontology and Prolog queries:

```
rosrun rosprolog rosprolog tutorial_reasoning
```

2) Test that the programmed queries work as expected:

```
comp_onTopTable(Obj,ValueNewObj).
comp_onTopTable('http://knowrob.org/kb/semRoom_semantic_map.owl#plate1', Value).
comp_onTopTable(Obj,'0.75').
```

3) Assert new knowledge in the ontology using the new prolog queries:

```
create_instance_from_class('http://knowrob.org/kb/semRoom_semantic_map.owl#cup','20',Int).
```

4) Assert the property of “aboveOfTable” to the new instance:

```
rdf_assert(semRoom_semantic_map:cup_20, knowrob:'aboveOfTable', '0.99').
```

5) Retrieve the objects that we just instantiate in our ontology:

```
rdf_triple(knowrob:'aboveOfTable',A, B).
```

Tutorial 7: System integration- Reasoning

In order to access the reasoning system from ROS package, we need to create a launch file to start the json_prolog node. Then, this node will load your ontology (OWL file) there.

Note: The package json_prolog is not an interactive shell like rosprolog, then you cannot access it directly. For this reason the launch file is needed with the following content:
// the package_name is a name of the package you want to integrate with knowrob and name.owl is a name of
//your OWL file.

```
<launch>
<param name="initial_package" type="string" value="package_name" />
<param name="initial_goal" type="string" value="owl_parse('package://knowrob_map_data/owl/name.owl')" />
<node name="json_prolog" pkg="json_prolog" type="json_prolog_node" cwd="node" output="screen" />
</launch>
```

You can find this launch file in your template folder.

Tutorial 7: System integration- Reasoning

Then, to query the Prolog predicates from your C++ program, you have to include the following lines in your code:

```
//include header file of json_prolog
#include <json_prolog/prolog.h>
using namespace json_prolog;

// initialize variable which will send queries to prolog
Prolog pl;

// send a query and write result to bdgs variable. Remember there is no DOT at the end of query.
PrologQueryProxy bdgs = pl.query("member(A, [1, 2, 3, 4]), B = ['x', A], C = foo(bar, A, B)");
```

Tutorial 7: System integration- Reasoning

If you want to print the results of the Prolog predicate in your C++ program, then you should include the next lines:

```
//the  bdg["A"] indicates the value assigned to the queried variable A. We iterate through all values, //because the variable A can have several of them.

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;
    cout << "C = " << bdg["C"] << endl;
  }
```

Take a look at the file “reasoningCpp.cpp”

Tutorial 7: System integration- Reasoning

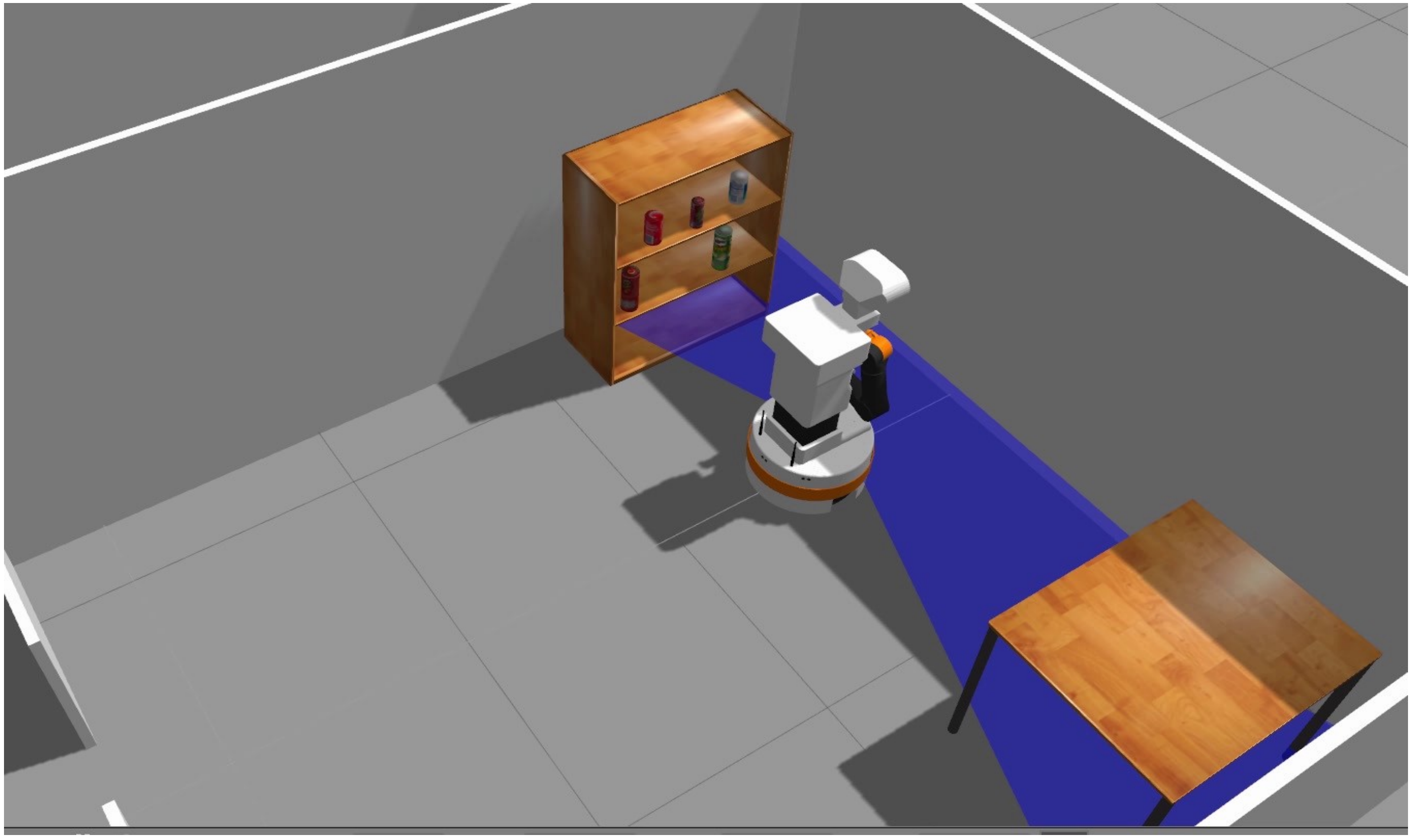
To run your ROS node that contains the reasoning and knowledge do:

1) Run the Json service
roslaunch tutorial_reasoning reasoning_cpp.launch

2) Run the node that contains the Prolog predicates
rosrun tutorial_reasoning reasoningInCpp_node

Chair for Cognitive Systems
Department of Electrical and Computer Engineering
Technische Universität München

Tutorial 7: System integration



Thank you

robocup.atHome.ics@tum.de

www.ics.ei.tum.de