Ubiquitous Semantics: Representing and Exploiting Knowledge, Geometry, and Language for Cognitive Robot Systems

Alexander Perzylo\textsuperscript{1}, Nikhil Somani\textsuperscript{1}, Stefan Profanter\textsuperscript{1}, Andre Gaschler\textsuperscript{1}
Sascha Griffiths\textsuperscript{2}, Markus Rickert\textsuperscript{1} and Alois Knoll\textsuperscript{3}

Abstract—In this paper, we present an integrated approach to knowledge representation for cognitive robots. We combine knowledge about robot tasks, interaction objects including their geometric shapes, the environment, and natural language in a common ontological description. This description is based on the Web Ontology Language (OWL) and allows to automatically link and interpret these different kinds of information. Semantic descriptions are shared between object detection and pose estimation, task-level manipulation skills, and human-friendly interfaces.

Through lifting the level of communication between the human operator and the robot system to an abstract level, we achieve more human-suitable interaction and thus a higher level of acceptance by the user. Furthermore, it increases the efficiency of communication.

The benefits of our approach are highlighted by examples from the domains of industrial assembly and service robotics.

I. INTRODUCTION

Knowledge representation for robotics is about connecting abstract representation with the “real world”. Moreover, if a robot is deployed in an environment in which it will encounter humans or even other autonomous robots it will have to have flexible representations which allow an alignment of its own representations with those of the agents around it.

One can call this approach {f ubiquitous semantics} which takes inspiration from the semantic web initiative. Using ontologies, one can tackle the problems which knowledge representation poses for modern robotics.

Ubiquitous semantics means that all relevant aspects of robot systems and their tasks are described in a way that preserves their inherent meaning. These semantic descriptions must be flexible and at a sufficiently generic level. This allows robots to share knowledge about how tasks are to be performed and completed. The descriptions are also flexible enough to describe the world in which the robot is moving but generic enough for a variety of environments and most importantly to allow for the non-deterministic nature of the environments in which robots are deployed, thus tackling the so-called “open world” problem. Also, such generic and flexible representations will be more amenable to the plasticity of human communication.

\textsuperscript{1}Alexander Perzylo, Nikhil Somani, Andre Gaschler, Stefan Profanter, and Markus Rickert are with fortiss GmbH, Guerickestr. 25, 80805 München, Germany \texttt{perzylo@fortiss.org}
\textsuperscript{2}Sascha Griffiths is with the Cognitive Science Research Group, School of Electronic Engineering and Computer Science, Queen Mary University of London, 10 Godward Square, London E1 4PF, United Kingdom
\textsuperscript{3}Alois Knoll is with the Department of Informatics VI, Technische Universität München, Boltzmannstr. 3, 85748 Garching, Germany

The remainder of the paper is structured in the following way. We will address the related work in the next section. This will be followed by a more general discussion of knowledge representation – specifically for robots. Against this background, we will discuss object detection, pose estimation, task execution, and human-friendly interfaces. We conclude with a few remarks on the general use of our ontology based knowledge framework.

II. RELATED WORK

Related work applies concepts from knowledge representation \textsuperscript{2}, symbolic task planning \textsuperscript{3}, and planning for natural language dialogs \textsuperscript{4}.

Many modern approaches of knowledge representation in robotics have taken the semantic web initiative as a source of inspiration. Those approaches make use of ontologies to organize knowledge in autonomous and intelligent systems.

The RoboEarth initiative \textsuperscript{5} makes use of this approach with the goal of achieving effective sharing of knowledge \textsuperscript{2}, data \textsuperscript{6}, and processing resources \textsuperscript{7} among robots. This is often referred to as cloud robotics, and has established advantages regarding memory and processing limits.
Additionally, models acquired by one robot can be re-used by another one.

There are other means by which robots can gain and apply knowledge. These can be categorized as “physical symbol grounding”, “grounding words in action” and “social symbol grounding” [8].

III. KNOWLEDGE REPRESENTATION

In order to endow robots with advanced cognitive capabilities, it is necessary to make all relevant aspects of their properties, tasks, and environment known to them. Encoding and interpreting knowledge about these different fields allows them to assess the applicability of their skills and to link their actions to a wider context.

In this section, we briefly summarize our methodology for semantically describing processes, related interaction objects, and their environment. We design a common description language based on the Web Ontology Language (OWL), which uses class taxonomies, instances of these classes, and properties for classes and instances.

A. Semantic process descriptions

Semantic process models are partially ordered sequences of tasks. Each type of task specifies its pre- and postconditions, and a set of parameters, of which some must be defined and others might be optional. An underspecified task can be fully parameterized through automatic reasoning, when a process model is assigned to a robot system, by combining the requirements of tasks with the capabilities of the selected system [1].

Fig. 1a depicts an excerpt of the semantic description of an industrial assembly process, which is visualized in Fig. 1b and Fig. 1c. It contains three tasks, i.e., AssembleBearingTreeTask, AssembleBearingPipeTask, and AssemblePipeTreeTask. Exemplarily, the associated object models for the AssemblePipeTreeTask are shown. The order of the tasks is given through PartialOrderingConstraints, which specify that the AssemblePipeTreeTask has to be executed after the other two tasks have been carried out.

B. Semantic environment description

Semantic environment descriptions encode the composition of physical entities of the real world, e.g., robots, tools, sensors, or tables, and abstract meta-information, e.g., available skills or environmental constraints [1].

The semantic description of the workcell in Fig. 2 specifies a robot, its base, a table, and an RGBD sensor. These entities are linked with the workcell instance FortissWorkcell through instances of type FixedJoint, e.g., robot-base_joint. The robot workspace is set to be constrained to the given cuboid (constraint C2), for which two subregions with different velocity limits have been defined (constraints C1 and C3).

C. Semantic object models

Next to basic object properties, e.g., type, name, weight, material, or bounding box, we are able to semantically
describe the geometric shape of objects using a boundary representation (BREP) [1], [9]. BREP preserves the exact mathematical models of contained curves and surfaces. This enables the system to define and interpret various geometric interrelational constraints, e.g., coincidence, concentricity, parallelity, etc., between two objects’ vertices, edges, or faces [9], [10].

Fig. 3 shows the BREP-based semantic description of a finite cylinder’s geometry. Selected correspondences between the visualization on the right and the ontological instances on the left are highlighted.

IV. OBJECT DETECTION AND POSE ESTIMATION

In this section, we present an approach for shape-based object detection and pose estimation based on semantic descriptions of object models. This involves deep object models that include exact information about the geometric properties of the object. This approach allows for the detection of symmetrical objects whose pose are inherently underspecified. Knowledge about sensor noise and manufacturing tolerances can also be explicitly included in the pose estimation step [11].

A. Geometric constraints from primitive shape matching

The object is modeled as a set of primitive shapes \( P \) (e.g. planes, cylinders) based on its boundary representation (BREP). Each primitive \( P_i \) enforces a set of constraints \( (C_{p_i}, C_{n_i}) \) on the position and orientation of the object respectively, where each row of \( C_{p_i} \) and \( C_{n_i} \) contains a direction along which the constraint has been set.

A complete set of primitive shapes is defined as a set where the constraints fully specify the 3D position and orientation of the object. A minimal set of primitive shapes is defined as a set which is complete but removing any primitive shape from the set would render it incomplete.

Table II presents the list of supported geometric constraints between primitive shapes, where

\[
\dot{p}_2 = Rp_2 + t, \quad \dot{p}_{21} = \dot{p}_2 - p_1, \quad \hat{n}_2 = Rn_2
\]

1) Feature Vectors for Sets of Primitive Shapes: Correspondences between the scene and model shape primitives are obtained by matching feature vectors constructed from geometric properties of the primitive shapes. These feature vectors not only encode the geometric properties of the shapes, but also of the relations between the shapes (see Table I). Minimal sets of primitives from the scene point cloud are calculated during the pose estimation stage (see Section IV-B.2), and the distance between the feature vectors provides a metric for obtaining hypotheses of shape associations.

B. Constraint Processing for incomplete pose estimation

1) Detection of minimal and complete sets of primitives: The constraints \( (C_{p_i}, C_{n_i}) \) enforced by each primitive shape \( P_i \) are stacked into two matrices \( C_p \) and \( C_n \) (each having 3 columns). The constraints are complete if the matrices \( C_p \) and \( C_n \) both have rank 3. Fig. 4b shows an example of a complete set of primitive shapes.

TABLE I: Feature vectors for primitive shape sets
\[
\begin{array}{|c|c|}
\hline
\text{Primitive shape} & \text{Feature Vector (fv)} \\
\hline
\text{Inf. Plane} & \phi \\
\text{Sphere} & \text{radius} \\
\text{Inf. Cylinder} & \text{radius} \\
\hline
\text{Plane+Plane} & \text{fv(plane1), fv(plane2), angle(plane1_normal, plane2_normal), min_distance(plane1, plane2)} \\
\hline
\text{Plane+Cylinder} & \text{fv(cylinder1), fv(plane), angle(plane_normal, cylinder_axis)} \\
\hline
\text{Cylinder+Cylinder} & \text{fv(cylinder1), fv(cylinder2), angle(cylinder1_axis, cylinder2_axis), min_distance(cylinder1, cylinder2)} \\
\hline
\text{Plane+Plane+Cylinder} & \text{fv(plane1, cylinder1), fv(plane2, cylinder2)} \\
\hline
\end{array}
\]

Algorithm 1 Detecting object poses using RANSAC

1: Input : \([P_s, [P_m]_{\text{min}}]\) (set of scene primitive shapes and minimal sets of model primitive shapes)
2: Output : \([T, s_{\text{max}}]\) (best pose estimate with score for detected object instance)
3: forall \( P_i \in [P_m]_{\text{min}} \)
4: \( s_{\text{max}} \leftarrow 0 \)
5: compute shape matching hypothesis (\( H_i \)) using fv’s, see Section IV-A.1
6: calculate transformation estimate \( T_i \) for \( H_i \), see Section IV-B.2
7: compute score \( s_i \) for hypothesis \( H_i \)
8: if \( s_i \geq \text{thresh} & s_i > s_{\text{max}} \)
9: \( T \leftarrow T_i \)
10: \( s_{\text{max}} \leftarrow s_i \)
11: EndFor

2) Constraint solving for pose estimation: The optimization is performed over transformations \( T \) that align the object model to the objects in the scene. The transformations are represented as \( \Delta x = (t, r) \) where \( t \) is the translation and \( r \) is the rotation in axis angle representation.

The optimization function is the absolute value of the transformation, i.e., minimization of \( ||\Delta x||_2 \). The constraint functions \( g_i \) along with their lower and upper bounds (\( \text{lb}(g_i), \text{ub}(g_i) \)) are obtained from the primitive shape matching constraints shown in Table II. The bounds \( (d_{\text{min}}, d_{\text{max}}) \) of the constraints can be used to incorporate the noise in sensor data or primitive shape fitting errors, as well as manufacturing uncertainties.

The resulting optimization problem is:

\[
\arg \min_{\Delta x} ||\Delta x||_2 \\
\text{subject to} \quad \text{lb}(g_i) \leq g_i \leq \text{ub}(g_i), \quad i = 1, \ldots, m.
\]

This set of equations is then solved using a non-linear least squares min-max solver (MA27) from [12] using the deterministic non-linear optimization utility from library Coin-OR (named IPOPT) [13]. If the constraints are complete, the pose is uniquely defined. Otherwise, the constraint solver returns one possible solution.
structures, a manipulation object serves as a useful reference for geometric relations, e.g. the $P \in \text{SE}(3)$ and a set of primitive shapes $\text{FK}$ kinematic function $R$ robot manipulators (including their tools) are defined by geometric constraints that relate objects and objects. Compared to the constraint resolution scheme in the object recognition component (Sec. IV-A), we perform a generic, iterative minimization of a cost function. For that, each constraint $C \in C$ is represented by a cost function $\text{Cost} : \text{SE}(3) \times \text{SE}(3) \mapsto \mathbb{R}^c$ that depends on the poses of two referenced shapes and returns a zero vector iff the constraint is fulfilled. To solve a given manipulation task, we minimize the stack of cost functions $q \in \mathbb{R}^n \mapsto \mathbb{R}^c$ and obtain a valid robot pose $q$. To ensure reliable convergence, cost functions are defined such that they are differentiable and reflect the correct number of $c$ constrained degrees-of-freedom [10].

Many robot tasks in manufacturing and service domains pose constraints on only a few degrees-of-freedom, while the remaining degrees-of-freedom can be used to fulfill qualitative, lower-priority goals. Such goals may include the avoidance of singularities or joint limits, waypoints close to a previous one for shorter trajectories, or distance maximization from obstacles. When cost functions $\text{Cost}$ allow computation of a full-rank Jacobian $J$, we can compute the null-space projection matrix $N$ of a task, $N(q) = 1 - J^\dagger(q)J(q)$, where $\dagger$ denotes the pseudo-inverse. Projecting a lower-priority control signal onto $N$ then allows null-space optimization of qualitative goals. As an example, the task of grasping a cylindrical object can semantically be defined by several coincidence constraints between a parallel gripper and the object. Based on these constraints, the robot will find a posture-optimized grasp along the rotational axes of the object.

VI. HUMAN-FRIENDLY INTERFACES

We aim at reducing the complexity of interacting with robot systems. But, relying solely on semantic descriptions would only shift the required expertise for using such systems from the field of robotics to the field of knowledge.

<table>
<thead>
<tr>
<th>Constraint ($i$)</th>
<th>Cost Function ($g_i$)</th>
<th>Bounds (lb, ub)</th>
<th>Constrained Spaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plane-Plane</td>
<td>$[n_i^T p_{21}; n_1^T n_1]$</td>
<td>lb : $[d_{\min}; a_{\min}]$</td>
<td>$C_n : [n_{1,1}; n_{1,2}]$</td>
</tr>
<tr>
<td>Cylinder-Cylinder</td>
<td>$[|p_{21} - (n_i^T p_{21})n_1|_2; n_i^T n_1]$</td>
<td>lb : $[d_{\min}; a_{\min}]$</td>
<td>$C_n : [n_{1,1}; n_{1,2}]$</td>
</tr>
<tr>
<td>Sphere-Sphere</td>
<td>$[p_{21}]$</td>
<td>lb : $a_{\min}$</td>
<td>$C_n : [n_{1,1}; n_{1,2}]$</td>
</tr>
</tbody>
</table>

Table II: Summary of supported constraints between primitive shapes.
our software framework. In the runtime resulting grammar can be shared between all robots using OpenCCG in the Wordnet class taxonomies of tasks and objects with links to concepts in the system’s ontologies. As a second step in this phase, an OpenCCG grammar is automatically generated [17], which serves as an input to our dialog component. The annotation has to be done only once for each type of task or object. The resulting grammar can be shared between all robots using our software framework. In the runtime phase, our dialog component uses the generated grammar to parse natural language input into a logical form, and to interpret it by mapping it back to concepts in the system’s ontologies.

1) Configuration Phase: Natural language utterances can be ambiguous. As a result, a naïve one-to-one mapping of an instruction verb to a type of task would likely fail. Preferably, all synonyms for a given verb or noun should be considered, when trying to interpret a command. For this reason, we annotate the classes in the task and object ontologies with Wordnet synonym sets (synset). Task classes are annotated with verb synsets, object classes with noun synsets, and classes that serve as discriminating features with adjective synsets.

For instance, the parameters objectToPick and objectToPlaceOn can be bound by selecting the desired object from a list, pointing at the object, or telling its name. This interface also supports the definition of assembly poses, grasp poses, and approach poses using geometric interrelational constraints [9], [10].

B. Natural Language Interface

This interface is not meant to support an open world dialog, but to instruct a robot system to perform a specific task. Interaction with our robot systems through natural language requires to map utterances to concepts in our ontologies, e.g., tasks and objects. We rely on a two-phases approach.

In the configuration phase, a human expert annotates the class taxonomies of tasks and objects with links to concepts in the Wordnet ontology. As a second step in this phase, an OpenCCG grammar is automatically generated [17], which serves as an input to our dialog component. The annotation has to be done only once for each type of task or object. The resulting grammar can be shared between all robots using our software framework. In the runtime phase, our dialog component uses the generated grammar to parse natural language input into a logical form, and to interpret it by mapping it back to concepts in the system’s ontologies.

1) Configuration Phase: Natural language utterances can be ambiguous. As a result, a naïve one-to-one mapping of an instruction verb to a type of task would likely fail. Preferably, all synonyms for a given verb or noun should be considered.

2) Runtime Phase: The OpenCCG grammar generated during the configuration phase is used by a dialog component to parse natural language utterances into a logical form. This representation is used to analyze a sentence’s structure, and how the different parts are semantically related to each other, e.g., which noun is the subject of which verb. Starting from the logical form, the robot system has to determine, which task the human operator intends to be executed.

This is achieved by grounding the sentence’s referents in OWL functional syntax linking the task type with a synonym set of associated verbs.

The grammar generation process takes an OpenCCG grammar template as an input. It contains the static parts of the grammar, i.e., functions, macros, and category definitions. The functions and macros are then used during the generation of the dynamic part of the grammar, e.g., to create the singular, singular third person, and plural forms of a verb. Furthermore, the template describes commonly used words which are not linked with concepts in our ontologies. For instance, definite and indefinite articles, prepositions, and pronouns. As a next step, the knowledge base is queried for all annotated task and object concepts, which results in a set of ontology concepts and their Wordnet synset annotations. The verbs, nouns, and adjectives from these synsets are then added to the grammar. An overview of the configuration phase in given in Fig. 6.

Fig. 6: Overview of configuration phase

Fig. 7 exemplarily shows the annotation of a service robot’s task description called ServeBeverage. It contains the AnnotationProperty linkedSynset, which links to a particular synset in the Wordnet ontology, i.e., synset-serve-verb-6.

Fig. 7: Excerpt of an annotated semantic task description in OWL functional syntax linking the task type with a synonym set of associated verbs.

The grammar generation process takes an OpenCCG grammar template as an input. It contains the static parts of the grammar, i.e., functions, macros, and category definitions. The functions and macros are then used during the generation of the dynamic part of the grammar, e.g., to create the singular, singular third person, and plural forms of a verb. Furthermore, the template describes commonly used words which are not linked with concepts in our ontologies. For instance, definite and indefinite articles, prepositions, and pronouns. As a next step, the knowledge base is queried for all annotated task and object concepts, which results in a set of ontology concepts and their Wordnet synset annotations. The verbs, nouns, and adjectives from these synsets are then added to the grammar. An overview of the configuration phase in given in Fig. 6.

2) Runtime Phase: The OpenCCG grammar generated during the configuration phase is used by a dialog component to parse natural language utterances into a logical form. This representation is used to analyze a sentence’s structure, and how the different parts are semantically related to each other, e.g., which noun is the subject of which verb. Starting from the logical form, the robot system has to determine, which task the human operator intends to be executed.

This is achieved by grounding the sentence’s referents in
the robot’s knowledge base. Verb phrases are considered to correspond to a task that shall be executed. They have different numbers of associated noun or prepositional phrases, which form their arguments. They refer to objects the tasks have to be performed upon. Hence, each argument has to be grounded in the robot’s knowledge base. The identification process first searches for all possible task candidates by matching the used verb with the synsets linked from the task concepts. This list is narrowed down by filtering out candidates, which require a different amount of arguments, or different types of arguments. If a single task could be identified, it is selected for execution, otherwise a disambiguation dialog is initiated [17]. The runtime phase is summarized in Fig. 8.

VII. CONCLUSION

In this paper, we show how to specify and execute abstract process descriptions and their tasks, e.g., using geometric interrelational constraints between involved objects to define an assembly or grasp pose. The representation of deep object models, which are required to formulate such constraints on individual edges or faces, is based on the BREP formalism. It encodes the exact geometric properties of the objects’ shapes. Using the knowledge on contained primitive shapes further improved the performance of our object detection and pose estimation.

In order to command the robot system through natural language, we automatically generate grammars to parse and map utterances to concepts in our ontological taxonomy of tasks and objects. Having described all relevant aspects of a robot system and its tasks in a semantic way (ubiquitous semantics), the system can benefit from synergy effects created through linking the available information and reasoning about its implications.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Union’s Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 287787 in the project SMErobotics, and FET grant agreement no. 611733 in the project ConCreTe.

REFERENCES


