An Ontology Framework for Physics-based Manipulation Planning

Mohammed Diab, Muhayyuddin, Aliakbar Akbari, and Jan Rosell *

Institute of Industrial and Control Engineering,
Universitat Politècnica de Catalunya, Barcelona, Spain,
{mohammed.diab,muhayyuddin.gillani,aliakbar.akbari,
jan.rosell}@upc.edu

Abstract. In manipulation planning, dynamic interactions between the objects and the robots play a significant role. In this scope, dynamic engines allow to consider them within motion planners, giving rise to physics-based motion planners that consider the purposeful manipulation of objects. In this context, the representation of knowledge regarding how the objects have to be manipulated eases a semantic-based reasoning that reduces the computational cost of physics-based planners. In this work, an ontology framework is proposed to organize the knowledge needed for physics-based manipulation planning, allowing to derive manipulation regions and behaviors. A semantic map is constructed to categorize and assign the manipulation constraints based on the robot, the objects and the type of actions. The ontology framework can be queried using Description Language to obtain the necessary knowledge for the robot to manipulate the objects in its environment.

1 Introduction

Manipulation planning in human environments is one of the challenging areas in robotics research. It is focused on making the robot capable of performing complex manipulation tasks, which requires manipulation planning capabilities in clutter and unstructured environments. These capabilities need the rich semantic description of the scene, knowledge about the physical behavior of the objects, and reasoning about the performed manipulation actions. Additionally, they require a detailed knowledge of the objects in the robot workspace and their physical properties (such as object dynamics during manipulation). Knowledge can be defined in the form of ontologies, that are a structured way of representing information, and is becoming a standardized way of representing the knowledge for the robots.

Various representation alternatives have been proposed to describe the knowledge about the way of manipulating objects. They define the relationships between the high-level description of the objects and the low-level data about the environment. To handle the physical behavior of the objects during manipulation, some approaches such as [1] are proposed that describe the physics-based knowledge and reason about it. These approaches are used to relate the physical quantities. For instance, the product of mass and

* This work was partially supported by the Spanish Government through the projects DPI2013-40882-P and DPI2016-80077-R.

acceleration describe the force (according to the Newton second law of motion). The main idea of these approaches is to formalize physics laws as logical axioms in an abstract model and derive what will happen from these axioms. However, the main issues are the predictions based on axioms of qualitative physics [1], because this formalization is complex and computationally intensive, particularly when manipulation actions are considered [2]. However, this problem can be avoided by using physics engines such as Open Dynamic Engine (ODE, http://www.ode.org/). These engines describe the dynamics of the system with precise accuracy, provided that the interaction dynamics, such as friction between the surfaces, bounciness of the surface and penetration depth of the rigid body are given. These parameters must be set carefully, since small changes may lead to different results.

The contribution of this paper is the proposal of an ontology framework for physics-based manipulation planning based on the formalization and extension of the ontology for physics-based manipulation planning proposed in [4]. In particular:

- *Formalization* according to the Ontology-based Unified Robot Knowledge OUR-K [5], a standardized ontological framework for service robots.
- *Semantic map extension* to automatically construct a semantic map to categorize the objects into different types according to the objects and task constraints.
- *Interaction dynamics extension* to define a knowledge that allows the planner to deal with interaction dynamics.

Description logic (DL) language will be used to describe the framework classes and to query the ontologies. The ontological framework copes with low-level information, i.e. physical parameters of objects, and high-level information. This framework aims to facilitate robots to perform manipulation tasks by providing, on the one hand, physical parameters to ODE simulation engine to accurately model the physical environment, and on other hand, knowledge describing how objects should be manipulated. No information regarding the uncertainty in the objects type or poses will be coded in the first version of the proposed ontology framework.

2 Related Work

Various studies have investigated the use of knowledge in the form of ontologies for the detailed description of real environments. Ontologies are usually designed by considering particular areas, such as task planning [6], space representation and navigation [7], locomotion [8], semantic representation for human collaboration [9], and others. However, many robotic tasks in real environments need task, motion and manipulation planning together. Therefore, the need for a generic and well-defined knowledge representation is becoming more evident.

Work on ontology standards for generalizing knowledge representation for robots has been done through the *Ontologies for Robotics and Automation Working Group* (ORA WG) [3, 10]. This group is divided into four sub-groups entitled: Upper Ontology/Methodology(UpOM), Autonomous Robots (AuR), Service Robots (SeR), and Industrial Robots (InR). Different standards are being developed in each sub-group, with CORA (Core Ontology for Robotics and Automation) being the common to all of them.

On the other hand, there are other efforts done to define ontologies frameworks for related fields, such as: KIEF [11] (Knowledge Intensive Engineering Framework) that describes a way to handle several activities of engineering, such as design, manufacturing, operation, maintenance, and recycling; KAON [12] (Karlsruhe ontology) that describes a generic ontological semantic structure that is used in OUR-K [5] to establish an ontology framework for service robots in human environment; the physics-based manipulation ontology in [13], involving knowledge about the world and the planning phase to address a high-level and a low-level reasoning processes related to task and motion planning; and the Open Semantic Framework [14] proposed to help cognitive robots to execute manipulation tasks by integrating ontology with a cloud-based engine used to detect the objects and retrieve their manipulation actions from the ontology.

2.1 Description of OUR-K Knowledge Classes

The main contribution of OUR-K [5] framework is the establishment of an ontology-based unified knowledge for service robots in human environments by integrating low-level data (sensory data) with high-level knowledge (context information). This framework is described within a concept hierarchy through the use of an ontology. It is composed of five main classes: feature, object, space, context, and action. Each knowledge class has three levels (except feature class that has two), and each level has three ontological layers: metaontology, ontology, and ontology instance. *Metaontology* is used to represent generic information, such as the concept of physical object in the service robotics field. *Ontology* is an ontology schema layer used for domain specific knowledge, for instance, in service robotics, ontology layer contains the knowledge of a particular domain such as kitchen. *Ontology instance* is used to store the information of the objects (such as their features). Because ontology is an object-oriented and frame-based language the metaontology layer can provide a template for ontology layer to build terminology, while the ontology instance layer can be defined as an individual frame. The information of ontological classes, properties, and instances is transferred within unidirectional reasoning in the same knowledge level. Whereas, bidirectional reasoning relates several knowledge classes or knowledge levels. OUR-K classes are described below:

Feature class is used to define how objects are perceived. It has two levels: perceptual feature and perceptual concept. In perceptual feature level, the real environment is described from the perceptual perspective, i.e. sensors, that perceive features of instances such as color, texture, and scale invariant feature transform (SIFT). In perceptual concept level, concepts are grounded to perceptual features.

Object class is used to define objects, their functionality and their parts. It is composed of three levels: part-object level, object level and compound level. Part-object includes parts of objects according to their functionality (for example, body and handle each has its own functionality, i.e. containing and grasping respectively). Object level includes object name and functionality, for instance, the composition of a *cup* is body and handle. In compound level, closely related objects that can be utilized together are linked (e.g. a *cup* and a *saucer*).

Space class builds a semantic map and uses it to enhance the representation of the environment, which requires geometrical, positional information and qualitative fea-

tures, to express relations. The construction of a semantic map depends on two types of maps: metric and topological. The former defines areas in the environment, which may be empty or occupied. The latter contains the information of the topology of the free region (e.g. with a graph extracted from a voronoi diagram). The relation between these maps and objects is stored in the semantic map.

Context class is used to understand the situation of the objects in the environment. This situation is derived from two types of relations: spatial and temporal, which are defined in spatial and temporal levels, respectively, such as *crowd*, which means that the robot will encounter some obstacles. The spatial level contains the spatial relations (such as *on, in, left, and right* functions) and space classes (semantic map). Temporal level contains the temporal relations such as *before, after, overlap, meet*.

Action class is used to define a task and the way of execution it. Action class consists of three levels: primitive behavior, sub-task and task. In primitive behavior, perceptual, motion and manipulation behaviors are defined (e.g. *turn, goto, extractcolor, extractSIFT*). In a sub-task level, a short-term sequence of behaviors are defined, such as *gotoSpace, localization*. A task such as *navigate, delivery* is defined in task level. It can be decomposed into sub-tasks, which can be decomposed into primitive behaviors. For example, *recognizeObject* is a task of finding an object within an image (e.g. *extractcolor, extractSIFT*).

2.2 First Proposal of a Physics-based Manipulation Ontology

In previous works [13], the authors proposed a manipulation ontology to represent knowledge to face the manipulation problems a motion planner has to deal with when the robot moves and encounters obstacles in the environment. Two classification of objects are proposed in this ontology, fixed bodies and manipulatable bodies (i.e. obstacles that can be pushed away). The former remains static during the whole planning process, even if collision happens with other manipulatable objects. The latter can be manipulated during the planning. The manipulatable bodies are classified as free manipulatable bodies and constrained-oriented manipulatable bodies. The free manipulatable bodies can move in any direction (according to the dynamics of rigid bodies) when collision occur. The constraint-oriented manipulatable bodies have some allowable motion directions and others are restricted. The manipulation constraints are modeled by defining the manipulatable region from where the object can be pushed from (i.e. where the robot should be located to apply a pushing force).

The knowledge is structured by defining six classes derived from a general manipulation class. These classes describe the initial state, goal state, action, region, object type, and object elements. Initial state class describes an initial location of the robot, while goal state class shows the final location of the objects. Action class contains different manipulation primitive actions, like pick, place and push. Region class defines three sub-classes: manipulation, object, and goal region. ManipulationRegion sub-class defines the region of manipulation, i.e. from where the robot can interact with the object. ObjectRegion sub-class is defined as a bounding box of an object. GoalRegion sub-class is a circular region defined around the goal state. ObjectType class defines the manipulatable objects and free objects and their constraints. ObjectElements describe the feature of the objects.

3 An Ontology Framework for Physics-based Manipulation

The representation of robot knowledge with an ontology has interesting properties like shareability and exchangeability [15]. Knowledge is represented by defining the properties and relations between concepts that provides a rich semantic description to aid the solving of particular problems. The aim of this paper is to modify the OUR-K ontology framework to cope with manipulation problems. As shown in Fig. 1, the proposed ontology framework has six classes. The object class and the context class are the same as in the OUR-K framework. The other four are described in the following subsections.

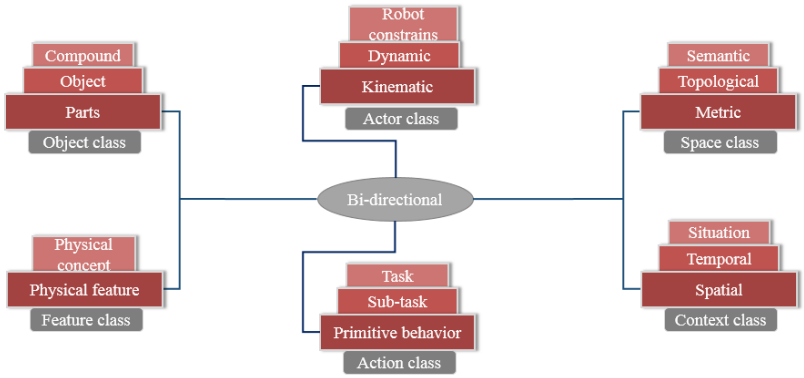

Fig. 1. Classes of the proposed physics-based ontologies for manipulation, based on the OUR-K framework.

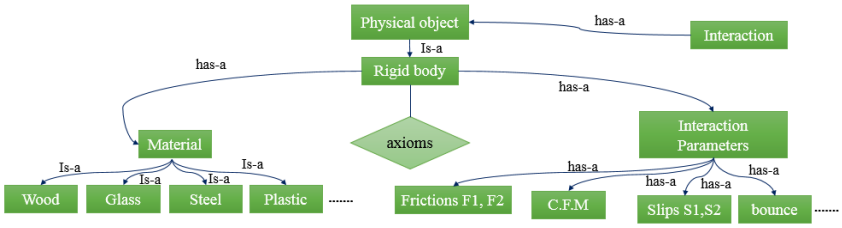

Fig. 2. Schema of feature class that explains the hierarchy of the concepts, features, and relations via axioms

Level	Tuple	Example
Feature level	Cp	*physicalFeature*, *InteractionParameter*, *material*, X , Y
	Rel	*has-physicalFeature*, *has-InteractionParameter*, *has-material*
	H^C	material: *physicalFeature*, X: material, Y: InteractionParameter
	H^R	has-material: *has-physicalFeature*
	A^0	$\exists X, Y$ | material(X) $\wedge$ InteractionParameter(Y) $\wedge$ has-*physicalFeature(X,Y)*
Concept level	Cp	*physicalObject*, *rigidBody*, *Interaction*, *X*
	Rel	*has-physicalObject*,*has-rigidBody*, *has-interaction*
	H^C	Rigid body: *physicalObject*, X: *Rigid body*
	H^R	has-Rigid body: *has-physicalObject*
	A^0	$\exists X, Y$ | Rigid body(X) $\wedge$ InteractionParameter(Y) $\wedge$ has-*InteractionParameter(X,Y)*

Table 1. Metaontology layer for the feature class. Cp, Rel, H^C, H^R, A^0 stand for concept, relation, hierarchy of concept, hierarchy of relation, and axioms, respectively, following nomenclature of [5].

3.1 Feature class

In the proposed framework, the feature class has two levels: physical concept and physical feature. The concepts of $rigidbody$ and $interaction$ are defined in the concept level, and their features in terms of material and interaction parameters in feature level. Fig. 2 describes the ontology schema of the feature class. For example, interaction is a concept defined as *"Contact between two surfaces"*, while its features depends on the type of material, i.e. each material has its properties like friction coefficients, density, etc. They are linked together (material and its properties) via axioms (facts), as described in Table1. For example, to handle a cup (which is made from A), the interaction occurs between two rigid bodies, cup and robot (its end-effector is made from B), and each has its physical properties, as described below using DL:

Interaction
$\wedge \exists hasSuperclass(Rigidbody, PhysicalObject)$
$\wedge \exists hasInterParameter(interactionParameter, Rigidbody)$
$\wedge \exists hasmaterial(material, Rigidbody)$
$\wedge \exists ismaterial(A, material)$
$\wedge \exists ismaterial(B, material)$
$\wedge \exists hasfeature(friction, interactionParameter)$

Axioms define that the material A has properties, such as friction coefficient, density, slip, CFM (Constrain Force Mixing), and bounce (see axioms of feature class in Table1). The physical features are used by the ODE simulator to accurately model the physical environment.

3.2 Actor class

Actor class is a new-defined class that describes the properties of the robot in the environment. It consists of three levels: robot kinematics, dynamics, and constraints.As described in Table 2, kinematic-level defines the kinematic structure of the robot and its location in the workspace, whereas dynamic-level contains the dynamic parameters of the joints of the robot, such as dumping, joint stiffness, and maximum efforts. In the constraints-level, based on kinematics and dynamics features, the robot working constraints are extracted.

Level	Tuple	Example
Kinematic level	Cp	*Robot, kinematic, forward, inverse, position, jointConfiguration, X,Y*
	Rel	*has-Robot*, *has-Kinematic*, *has-forward*, *has-inverse*, *has-position*, *has-jointConfiguration*
	H^C	forward: *position*, inverse: *jointConfiguration*, X: Robot, Y: Kinematic
	H^R	has-forward: *has-position*, has-inverse: *has-jointConfiguration*
	A^0	$\exists X, Y \mid$ Robot(X) $\wedge$ Kinematic(Y) $\wedge$ jointLimitations(Y)
Dynamic level	Cp	*Dynamic*, *DynamicParameter*
	Rel	*has-Dynamic*, *has-DynamicParameter*
	H^C	Dynamic: DynamicParameter
	H^R	has-Dynamic: *has-DynamicParameter*
	A^0	
Robot Constraint	Cp	*RobotConstraint, jointConstraint, DynamicConstraint, X, Y*
	Rel	*has-RobotConstraint*, *has-jointConstraint*, *has-DynamicConstraint*
	H^C	RobotConstraint: *jointConstraint*, RobotConstraint: *DynamicConstraint* Y: RobotConstraint
	H^R	has-jointConstraint: *has-RobotConstraint*, has-DynamicConstraint: *has-RobotConstraint*
	A^0	$\exists X, Y \mid$ Robot(X) $\wedge$ RobotConstraint(Y) $\wedge$ *has-RobotConstraint(X,Y)*

Table 2. Metaontology layer for the actor class.

3.3 Space class

figures/SemanticClass-eps-converted-to.pdf

Fig. 3. Schema of space class that describes the outcome of semantic map. The legend of different color shows that the classes information that is required to generate the semantic map.

In our framework, the space class contains three knowledge levels: metric map, topological map, and semantic map. As described in Table 3, a metric map contains empty and occupied spaces (by either objects or the robot). Rigid body is classified into single, and composite (see Fig. 3). Topological map defines the topology of the workspace, including where the objects are located in the workspace. Semantic map categorizes physical objects in the environment along the manipulation constraints. These constraints are defined as a function of the robot. It means that semantic map S depends on four parameters $S = (O, R, A, T)$, where: O is the set of objects that is provided by object class, R is the set of robot working constraints that is provided by the actor class, A is an action that is provided by the action class, and T is a topological map that is provided by the space class. The outcome of the semantic map is to assign the manipulation constraint by using DL reasoning, i.e. as shown in Fig. 3, the outcome of the semantic map due to reasoning is the type of object and its constraints.

3.4 Action class

The action knowledge class is composed of three levels: primitive behavior, sub-task, and task levels. A task is decomposed into sub-tasks, while sub-tasks involve primitive

Level	Tuple	Example
Metric map	Cp	*MetricMap, Space, EmptySpace, OccupiedSpace*
	Rel	*has-MetricMap, has-EmptySpace, has-OccupiedSpace*
	H^C	EmptySpace: *Space*, OccupiedSpace: *Space*
	H^R	has-EmptySpace: *has-Space*, has-OccupiedSpace: *has-Space*
	A^0	
Topological map	Cp	*TopologicalMap, singleObject, compositeObject*, X, Y
	Rel	*has-TopologicalMap,has-rigidBody, has-singleObject, has-compositeObject, has-Robot*
	H^C	TopologicalMap: *WorkSpace*, rigidBody: *singleObject*, rigidBody: *compositeObject*, TopologicalMap: *Robot*
	H^R	has-singleObject: *has-TopologicalMap*, has-compositeObject: *has-TopologicalMap*, has-Robot: *has-TopologicalMap*
	A^0	$\exists X, Y \mid$ singleObject(X) $\wedge$ OccupiedSpace(Y) $\wedge$ *has-OccupiedSpace(X,Y)* $\exists X, Y \mid$ compositeObject(X) $\wedge$ OccupiedSpace(Y) $\wedge$ *has-OccupiedSpace(X,Y)* $\exists X, Y \mid$ Robot(X) $\wedge$ OccupiedSpace(Y) $\wedge$ *has-OccupiedSpace(X,Y)*
Semantic map	Cp	*SemanticMap, rigidBody, RobotConstraint, action, X, Y*
	Rel	*has-SemanticMap: has-rigidBody has-SemanticMap: has-RobotConstraint, has-SemanticMap: has-action*
	H^C	rigidBody: *SemanticMap*, RobotConstraint: *SemanticMap*
	H^R	
	A^0	$\exists X, Y \mid$ SemanticMap(X) $\wedge$ action(Y) $\wedge$ *has-OccupiedSpace(X,Y)*

Table 3. Metaontology layer for the space class.

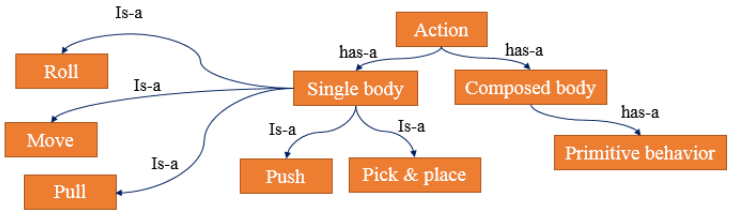

Fig. 4. Schema of action class that is divided into two sub-classes: single and composed body. Some of the actions are mentioned in each sub-class.

behaviors. Table 4 explains the function of the three levels. X and Y could be one or a set of actions, as shown in Fig. 4. The ontology schema of the layers of the three levels in action class is instantiated by a planner. When requested to plan, the planner checks on the topological map to know which object (or robot) is occupied with which space. Then, the semantic map is used to extract the constraints of the object (single or composite), and the robot w.r.t the action. Context class, particularly temporal level, can be used by the planner to set the sequence of action of a task.

4 Usage

To illustrate the proposal some simulation examples are performed using The Kautham Project [**?**]. It is a C++ based open-source tool for motion planning, that includes geometric, kinodynamic, and physics-based motion planner. It uses OMPL [**?**] for the core set of planning algorithm. OMPL is a C++ based open-source library for sampling based motion planning. Physics-based simulation are modeled using Open Dynamic Engine (ODE) that is used as state propagator for physics-based planning algorithm.

Level	Tuple	Example
PrimitiveBehavior	Cp	*PrimitiveBehavior, PhysicalBehavior, MotionBehavior, ManipulationBehavior, X, Y*
	Rel	*has-Behavior*, *has-BehaviorTarget*, *has-physicalParameter*
	H^C	PrimitiveBehavior: *PhysicalBehavior*, PrimitiveBehavior: *MotionBehavior*, PrimitiveBehavior: *ManipulationBehavior*
	H^R	
	A^0	$\exists X, Y$ | PrimitiveBehavior(X) $\wedge$ object(Y) $\wedge$ *has-BehaviorTarget(X,Y)*
SubTask level	Cp	*SubTask*, *MotionSubTask*, *ManipulationSubTask*, *X*, *Y*
	Rel	*has-SubTask*,*has-SubTaskTarget*, *has-SubTaskPhysicalParameter*, *has-Sequence*
	H^C	SubTask: *MotionSubTask* SubTask: *ManipulationSubTask*
	H^R	has-Behavior: *has-SubTask*
	A^0	$\exists X$,$\exists Y$ | SubTask(X) $\wedge$ Behavior(Y) $\wedge$ *has-Sequence(X,Y)*
Task level	Cp	*Task, X, Y*
	Rel	*has-Task*, *has-TaskTarget*, *has-TaskPhysicalParameter*
	H^C	
	H^R	has-SubTask: *has-Task*
	A^0	$\exists X$,$\exists Y$ | Task(X) $\wedge$ SubTask(Y) $\wedge$ *has-SubTask(X,Y)*

Table 4. Metaontology layer for the action class.

Ontologies are encoded using the Web Ontology Language (OWL) [**?**]. XML-based file format is used to access and share such knowledge. The ontologies are designed using the Protégé (http://protege.stanford.edu/) editor. It is an open-source software that can provide an ontology editor to develop knowledge-based applications. The query over the ontological knowledge is performed using Prolog predicates. A sequence of

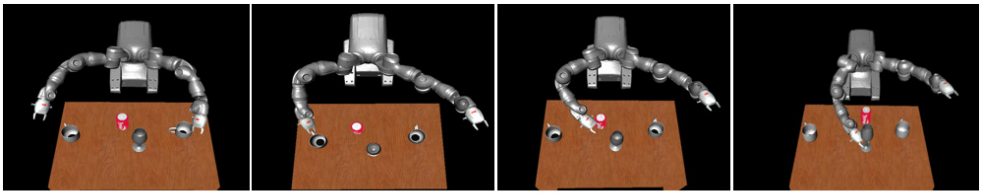

Fig. 5. Kautham scenes that explain the scenarios of manipulation the cup and wine-glass: a) push the cup, b) pick the cup, c) move the obstacle coca-can then, d) grasp the wine-glass.

actions was computed using the task planning approach proposed in [13]. In order to know the way of executing these actions, ontological knowledge can be used. A high level task such as $organizethetable$ can be satisfied by applying some actions like push, pick, place, grasp, etc. In the following examples, some queries to the ontology framework are presented using DL.

Fig. 5 shows a scene of The Kautham Project. It has two types of objects: cup (body and handle), that is a pickable or pushable object, and a Coca-can that is a pushable object. The aim is to show the manipulation behavior of the objects with respect to the constraints of the robot, the objects and the type of action. For example, the cup

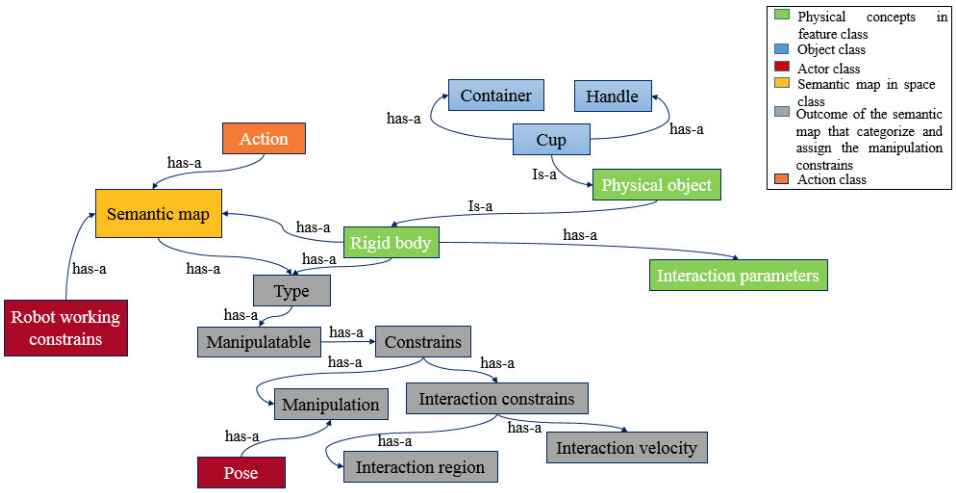

Fig. 6. Example of the cup that shows the required information from classes to apply pick and push actions. The legend defines the name of classes.

is a pickable object using its handle, or a pushable object using the body, as shown in example (1) and (2) below. Fig. 6 depicts the flow of information between the classes.

Additionally, in example (3), the behavior to grasp the wine-glass, that can be done from the cylinder (leg) (according to robot gripper constraint with respect to the diameter of the wine-glass) is described. As shown in Fig. 5, the path of grasping should be cleared (i.e. removing the obstacle of Coca-can). There are many scenarios to remove the obstacles, in order to grasp the goal object (wine-glass), such as pick, push or pull. However, with respect to the robot and object constraints, pick and place actions from the body of wine-glass are not possible in this case. In addition, regarding the information in the context class *"Crowd"* (that means the robot will encounter some obstacles), the best scenario is manipulation path, as a sub-task provided by the task planner (i.e. a push action is applied to move the Coca-can to execute the grasp action of the wine-glass).

Example 1: This example describes the query to the knowledge to push a cup.

Cup:= rigid body
$\wedge \exists hasSuperclass(PhysicalObject, Rigidbody)$
$\wedge \exists hasPart(Cup, handle)$
$\wedge \exists hasPart(Cup, body)$
$\wedge \exists hasAction(Cup, push)$
$\wedge \exists hasType(Cup, manipulatable)$
$\wedge \exists hasInteractionRegion(Cup, body)$
$\wedge \neg \exists hasInteractionRegion(Cup, handle)$
$\wedge \exists hasInteractionVelocity(body, velocity)$
$\wedge \exists hasInteractionParameter(body, physicalProperties)$
$\wedge \exists hasInteractionParameter(gripper, physicalProperties)$

Example 2: This example describes the query to the knowledge to pick a cup.

Cup:= rigid body
$\wedge \exists hasSuperclass(PhysicalObject, Rigidbody)$

$\wedge\exists hasPart(Cup, handle)$
$\wedge\exists hasPart(Cup, body)$
$\wedge\exists hasAction(Cup, pick)$
$\wedge\exists hasType(Cup, manipulatable)$
$\wedge\exists hasInteractionRegion(Cup, handle)$
$\wedge\neg\exists hasInteractionRegion(Cup, body)$
$\wedge\exists hasInteractionVelocity(body, velocity)$
$\wedge\exists hasInteractionParameter(handle, physicalProperties)$
$\wedge\exists hasInteractionParameter(gripper, physicalProperties)$

Example 3: This example describes two queries to the knowledge to pick the wine-glass when it is obstructed by an obstacle (Coca-can), set by the task planner.

a. Push obstacle Coca-can:= rigid body
$\wedge\exists hasSuperclass(PhysicalObject, Rigidbody)$
$\wedge\exists hasObstacles(Rigidbody, Cocacan)$
$\wedge\exists hasAction(Cocacan, push)$
$\wedge\exists hasType(Cocacan, manipulatable)$
$\wedge\exists hasInteractionRegion(Cocacan, body)$
$\wedge\exists hasInteractionVelocity(body, velocity)$
$\wedge\exists hasInteractionParameter(body, physicalProperties)$
$\wedge\exists hasInteractionParameter(gripper, physicalProperties)$

b. Grasp the wine-glass
Wine-glass:= rigid body
$\wedge\exists hasSuperclass(PhysicalObject, Rigidbody)$
$\wedge\exists hasObject(Rigidbody, Wineglass)$
$\wedge\exists hasAction(Wineglass, grasp)$
$\wedge\exists hasType(Wineglass, manipulatable)$
$\wedge\exists hasInteractionRegion(Wineglass, cylinder)$
$\wedge\exists hasInteractionVelocity(cylinder, velocity)$
$\wedge\exists hasInteractionParameter(cylinder, physicalProperties)$
$\wedge\exists hasInteractionParameter(gripper, physicalProperties)$

5 Conclusion

In this paper, we presented a knowledge representation in terms of ontologies for physics-based manipulation planning. Instead of making predictions based on inferences, a semantic map is generated to categorize and assign the manipulation constraints using reasoning based on logical axioms. We extended the OUR-K framework with our previous ontology for physics-based manipulation problems, by adding actor class, which describes the robot working constraints. The three examples of $pick$, $push$ the cup and grasp the wine-glass (after removing the obstacle) are successfully executed using The Kautham Project integrated with ODE dynamic engine. This ontology formulation is currently being extended with perception features, concepts and procedures.

References

1. Weld, D.S., De Kleer, J.: Readings in qualitative reasoning about physical systems. Morgan Kaufmann (2013)
2. Kunze, L., Dolha, M.E., Guzman, E., Beetz, M.: Simulation-based temporal projection of everyday robot object manipulation. In: The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1, International Foundation for Autonomous Agents and Multiagent Systems (2011) 107–114

3. Robotics, I., Society, A.: Ieee standard ontologies for robotics and automation. IEEE (April 2015) 1–60
4. Ud Din, M., Akbari, A., Rosell Gratacòs, J.: Ontological physics-based motion planning for manipulation. In: Proceedings of the 20th IEEE International Conference on Emerging Technologies and Factory Automation, Institute of Electrical and Electronics Engineers (IEEE) (2015)
5. Lim, G.H., Suh, I.H., Suh, H.: Ontology-based unified robot knowledge for service robots in indoor environments. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans **41**(3) (2011) 492–509
6. Cambon, S., Alami, R., Gravot, F.: A hybrid approach to intricate motion, manipulation and task planning. The International Journal of Robotics Research **28**(1) (2009) 104–126
7. Bateman, J.A., Farrar, S.: Modelling models of robot navigation using formal spatial ontology. In: Spatial Cognition, Springer (2004) 366–389
8. Chatterjee, R., Takao, I., Matsuno, F., Tadokoro, S.: Robot description ontology and bases for surface locomotion evaluation. In: Safety, Security and Rescue Robotics, Workshop, 2005 IEEE International, IEEE (2005) 242–247
9. Mozos, O.M., Triebel, R., Jensfelt, P., Rottmann, A., Burgard, W.: Supervised semantic labeling of places using information extracted from sensor data. Robotics and Autonomous Systems **55**(5) (2007) 391–402
10. Fiorini, S.R., Bermejo-Alonso, J., Gonçalves, P., de Freitas, E.P., Alarcos, A.O., Olszewska, J.I., Prestes, E., Schlenoff, C., Ragavan, S.V., Redfield, S., et al.: A suite of ontologies for robotics and automation [industrial activities]. IEEE Robotics & Automation Magazine **24**(1) (2017) 8–11
11. Yoshioka, M., Umeda, Y., Takeda, H., Shimomura, Y., Nomaguchi, Y., Tomiyama, T.: Physical concept ontology for the knowledge intensive engineering framework. Advanced Engineering Informatics **18**(2) (2004) 95–113
12. Bozsak, E., Ehrig, M., Handschuh, S., Hotho, A., Maedche, A., Motik, B., Oberle, D., Schmitz, C., Staab, S., Stojanovic, L., et al.: Kaontowards a large scale semantic web. E-Commerce and Web Technologies (2002) 231–248
13. Akbari, A., Gillani, M., Rosell, J.: Reasoning-based evaluation of manipulation actions for efficient task planning. In: Robot 2015: Second Iberian Robotics Conference, Springer (2016) 69–80
14. Tosello, E., Fan, Z., Castro, A.G., Pagello, E. In: Cloud-Based Task Planning for Smart Robots. Springer International Publishing, Cham (2017) 285–300
15. Alirezaie, M.: Bridging the Semantic Gap between Sensor Data and Ontological Knowledge. PhD thesis, Örebro university (2015)