A Tool for Knowledge-oriented Physics-based Motion Planning and Simulation

Muhayyuddin[1], Aliakbar Akbari[1], Jan Rosell[1⋆], and Wajahat Mahmood Qazi[2]

[1] Institute of Industrial and Control Engineering,
Universitat Politècnica de Catalunya, Barcelona, Spain,
{muhayyuddin.gillani,aliakbar.akbari,jan.rosell}@upc.edu
[2] Department of Computer Science,
COMSATS Institute of Information Technology, Lahore, Pakistan,
wmqazi@ciitlahore.edu.pk

Abstract. The recent advancements in robotic systems set new challenges for robotic simulation software, particularly for planning. It requires the realistic behavior of the robots and the objects in the simulation environment by incorporating their dynamics. Furthermore, it requires the capability of reasoning about the action effects. To cope with these challenges, this study proposes an open-source simulation tool for knowledge-oriented physics-based motion planning by extending *The Kautham Project*, a C++ based open-source simulation tool for motion planning. The proposed simulation tool provides a flexible way to incorporate the physics, knowledge and reasoning in planning process. Moreover, it provides ROS-based interface to handle the manipulation actions (such as push/pull) and an easy way to communicate with the real robots.

1 Introduction

Planning and simulation play an important role in robotics research. These are essential tools for the development of strategies and algorithms in various areas of robotics such as motion planning, grasping and manipulation. Moreover, these tools allow to demonstrate the proposed strategies under different environmental conditions and constraints. The existing software for robotics can be classified into two categories *single domain* and *multi domain* software. The former are designed to address the problem in a specific domain of robotics. For instance, *GraspitIt!* [1] is developed to study the grasp planning problems, while *MoveIt!* [2], *Robotic Library* [3] and *OpenRave* [4] are used to study the motion planning issues. On the contrary, the latter are designed in a generalized way to study the multiple domain problems, such as *Simox* [5] that is designed to study motion planning and grasping or *The kautham Project* [6] that is used to study task and motion planning.

To capture the realistic behavior in simulation, it is required to incorporate the dynamics of the robot and the environment. Simulation of dynamics is a challenging task due to the fact that robotic systems are nonlinear in nature and due to the difficulty in

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

determining the exact values of the parameters involved (such as forces that are acting on the system). To handle these issues the use of physics engines (such as ODE, www.ode.org and Bullet, bulletphysics.org) in robotic simulations is becoming popular. These engines provide a good approximation of rigid body dynamics. Moreover, physics engines are also used to develop the dynamic simulators, such as Gazebo (gazebosim.org) which provide a dynamic simulation environment for robotics. Beside the core robotic software, various middle-ware frameworks (such as ROS [7] and OROCOS [8]) are proposed to manage the communication between simulation and robot hardware. These middle-wares help greatly to simplify interprocess communications and synchronization issues.

The increasing complexity in the robotic systems, such as those including collaborative robots (new generation of industrial robots) or humanoid robots, set new challenges for robotic software. These challenges involve the rich semantic description of the environment for the understanding of the scene, the incorporation of dynamics in planning, the capability of reasoning about the performed actions, and computational efficiency. It is difficult to find a software that addresses all these challenging issues. The current study contributes along this line and proposes a simulation framework for knowledge-oriented physics-based motion planning. The current proposal extends *The Kautham Project* by integrating the ontological knowledge, reasoning and physics in the planning process.

The rest of the paper is structured as follows. The proposed framework is explained in Sec. 2. It involves the summary of the dependencies of the proposed simulation framework, a brief overview of *The Kautham Project*, and implementation details of how physics, knowledge and reasoning in planning process have been incorporated. Finally Sec. 3 concludes the study.

2 Knowledge-Oriented Physics-based Planning Framework

The proposed framework (Fig. 1) is developed by extending *The Kautham Project*. This section will briefly explain *The Kautham Project* and the implementation details of the proposed extensions for incorporating knowledge, reasoning and physics in planning.

2.1 Dependencies

The robotic simulation software depends on several concepts such as robot modeling, 3D rendering and collision checking. It is difficult to develop a standalone application from scratch and therefore, in order to incorporate such features, usually already existing libraries are used. The major dependencies of Knowledge-oriented physics-based planning framework are those of *The Kautham Project*. It is developed using C++ and uses many features of C++11 such as *std* features. It uses the CMake (www.cmake.org) build system and it is available under GIT (git-scm.com) version control system, and can be downloaded from sir.upc.ed/kautham. The GUI is designed in Qt (qt-project.org), 3D rendering is performed using Coin3D (www.coin3d.org). Robots and obstacles are defined using the Unified Robotic Description Format (URDF, http://wiki.ros.org/urdf), and the Eigen library (http://eigen.tuxfamily.org/) is used for linear algebra. The Open Motion Planning library (OMPL [9]) is used as planning core. It provides various sampling-based motion planners such as RRT [10], PRM [11] and KPIECE [12]. Moreover, it also

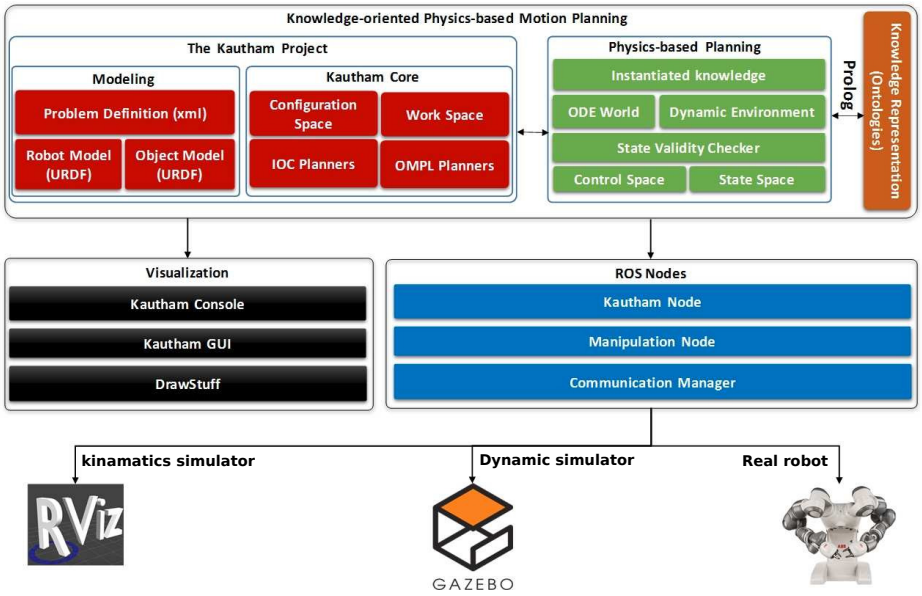

Fig. 1: Simulation framework for knowledge-oriented physics-based motion planning.

provides the capability of planning in state space where ODE is used as state propagator. The current proposal enhances the use of ODE in planning process to incorporate physics using knowledge-based reasoning. The knowledge is represented using Web Ontology Language (OWL [13]) and the Prolog language (http://www.swi-prolog.org/) is used for the reasoning over knowledge.

2.2 The Kautham Project

The Kautham Project is a C++ based open-source software for motion planning. It is used for teaching and research purposes at the Institute of Industrial and Control Engineering (IOC-UPC). For research, it is used to develop and demonstrate the motion planning algorithms, particularly for mobile and dexterous manipulators (arms equipped with anthromorphic hands and a mobile base).

Modeling: A motion planning problem is described using an XML file (Fig. 2). It consists of four components: robot model, object model, controls and planner. The main parameters that are specified for robots/objects are the path to the corresponding model, translation limits (in case of mobile base) and initial position with respect to the world frame. Controls are used to define the way how the degree-of-freedoms (dof) will be actuated. In the simplest case, one control per dof is considered. Controls can also be specified for obstacles (detailed explanation of controls can be found in [6]). The final part of the problem XML file specifies the name of the planner used (such as RRT), the planning parameters (such as planning time and goal bias) and the query that contains the start and the goal configurations.

```
<?xml version="1.0"?>
<Problem name="RRTYumi" topology="SE3">
   <Robot robot="robots/OpenDERobots/yumi.urdf" scale="1">
        <Limits name="X" min="-2.0" max="2.0" />
        <Limits name="Y" min="-2.0" max="2.0" />
        <Limits name="Z" min="-2.0" max="2.0" />
        <Home TH="0.0" WZ="1.0" WY="0.0" WX="0.0" Z="2.0" Y="0.0" X="0.0" />
   </Robot>
   <Obstacle obstacle="obstacles/table.urdf" scale="1">
        <Home TH="0.0" WZ="0.0" WY="0.0" WX="1.0" Z="2.0" Y="0.0" X="0.0" />
   </Obstacle>
   <Controls robot="controls/yumi.cntr"/>
   <Planner>
        <Parameters>
        <Name>RRTYumiPlanner</Name>
        <Parameter name="Constraint Force Mixing">0.3000000119</Parameter>
        <Parameter name="Control Dimensions">7</Parameter>
        <Parameter name="Error Reduction Parameter">0.5</Parameter>
        <Parameter name="Goal Bias">0.0500000003</Parameter>
        <Parameter name="Max Contacts">3</Parameter>
        <Parameter name="Max Control Steps">30</Parameter>
        <Parameter name="Max Planning Time">40</Parameter>
        <Parameter name="Min Control Steps">5</Parameter>
        <Parameter name="PropagationStepSize">0.02</Parameter>
        </Parameters>
        <Query>
          <Init dim="7">0.500 0.767 0.500 0.502 0.500 0.389 0.500</Init>
          <Goal dim="7">0.5 0.5 0.5 0.5 0.518 0.457 0.086 </Goal>
        </Query>
    </Planner>
</Problem>
```

(a)

```
iml version="1" encoding="UTF-8"?>
ntrolSet>

     <Offset>
             <DOF name="yumi/link_1_r" value="0.500"/>
             <DOF name="yumi/link_2_r" value="0.500"/>
             <DOF name="yumi/link_3_r" value="0.500"/>
             <DOF name="yumi/link_4_r" value="0.500"/>
             <DOF name="yumi/link_5_r" value="0.500"/>
             <DOF name="yumi/link_6_r" value="0.500"/>
             <DOF name="yumi/link_7_r" value="0.500"/>
     </Offset>
     <Control name="R/Shoulder1" eigValue="1">
              <DOF name="yumi/link_1_r" value="1"/>
     </Control>
     <Control name="R/Shoulder2" eigValue="1">
              <DOF name="yumi/link_2_r" value="1"/>
     </Control>
     <Control name="R/Shoulder3" eigValue="1">
             <DOF name="yumi/link_3_r" value="1"/>
     </Control>
     <Control name="R/Elbow" eigValue="1">
             <DOF name="yumi/link_4_r" value="1"/>
     </Control>
     <Control name="R/Wrist1" eigValue="1">
             <DOF name="yumi/link_5_r" value="1"/>
     </Control>
     <Control name="R/Wrist2" eigValue="1">
             <DOF name="yumi/link_6_r" value="1"/>
     </Control>
     <Control name="R/Wrist3" eigValue="1">
             <DOF name="yumi/link_7_r" value="1"/>
     </Control>
:ontrolSet>
```

(b)

Fig. 2: (a) an example of a problem file. (b) an example of a control file.

A robot is defined as a kinematic tree with optional mobile base, its configuration space is $R = SE(3) \times \mathbb{R}^n$, where n represents the number of joints of the robot. In case of fixed base, the $SE(3)$ part is represented as null. The kinematic structure of the robot is defined using URDF (Fig. 3). It contains the visual robot model, the collision model, transformations between the links, joints (along with limits) and dynamic parameters such as damping and masses. The visualization model is defined with triangular meshes that can be represented in *.wrl*, *.stl* and *.dae* formats. The collision model can be represented either by a triangular mesh or by primitive shapes (cylinder, box and sphere). Obstacles are also defined as robot data structures, in case of fixed obstacles, none of its dof are actuated.

Kautham-core: It consists of the workspace, the configuration space and a set of planners. Once a problem is loaded, it fills the data structures of the workspace (that includes the robot/obstacle models, their kinematic limits), and of the configuration space. Moreover, it contains the methods for collision checking using PQP [14] or FCL [15], and forward kinematics to move the robot to the particular configuration. To sample the configuration space various state samplers (such as random, Gaussian and Halton) are included.

Two families of planning algorithms are implemented in *The Kautham Project*, IOC planners and OMPL planners. The former contains potential field-based planners using navigation functions [16] and harmonic functions [17]. The latter contains the sampling-based geometric and kinodynamic planners (such as RRT, PRM and EST) offered by OMPL. The detailed explanation regarding the implementation of planners can be found here https://sir.upc.edu/projects/kautham/.

```
<link name="link_1_r">
   <inertial>
     <origin xyz="0 -0.03 0.12"/>
     <mass value="2"/>
     <inertia ixx="0.1"  ixy="0"  ixz="0" iyy="0.1" iyz="0" izz="0.1" />
   </inertial>
   <visual>
     <geometry>
       <mesh filename="YuMi/link_1.stl"/>
     </geometry>
   </visual>
   <collision>
     <origin xyz="-0.003 -0.023 0.069" rpy="1 0.44 0"/>
     <geometry>
       <box size="0.075 0.082 0.08"/>
     </geometry>
   </collision>
 </link>

 <joint name="joint_1_r" type="revolute">
   <parent link="body"/>
   <child link="link_1_r"/>
   <origin xyz="0.05355 -0.0725 0.41492" rpy="-0.9781 -0.5716 -2.3180"/>
   <axis xyz="0 0 1"/>
 <limit effort="300.0" lower="-2.9394" upper="2.9394" velocity="3.14"/>
   <dynamics damping="0.5"/>
 </joint>
```

(a)

```
<?xml version="1.0"?>
<robot name="table">
   <link name="base">
   <inertial>
         <origin xyz="0 0 0" rpy="0 0 0"/>
         <mass value="3"/>
         <inertia ixx="0.0683333"  ixy="0"  ixz="0"
                  iyy="0.0683333" iyz="0" izz="0.0683333"/>
   </inertial>
   <visual>
        <origin xyz="0 0 -0.028" rpy="0 0 0" />
        <geometry>
         <mesh filename="tablewood.dae"/>
        </geometry>
    </visual>
    <collision>
         <origin xyz="0 0 0" rpy="0 0 0" />
         <geometry>
                <box size="0.1399 0.0988 0.05586" />
         </geometry>
         <material>
                <color rgba="0.5 0.5 0.5 1" />
         </material>
        </collision>
   </link>
</robot>
```

(b)

Fig. 3: (a) a part of the URDF file of the robot. (b) the URDF model of the table

2.3 Physics-Based Planning

Physics-based motion planners have recently emerged as an extension to the kinodynamic motion planners, in which the robot can interact with the objects in the environment to purposefully manipulate. These interactions are modeled using rigid-body dynamics. The tree-based kinodynamic motion planners can easily be extended for physics-based planning by replacing the state propagator with dynamic engines (such as ODE). Moreover, the extension requires the proper definition of the state validity checker, the contact dynamics and the control space.

To enable the physics-based planning, ODE is used to handle the rigid body dynamics during propagation. It is an open-source C++ based dynamic engine widely used in the robotics community. Moreover, OMPL provides a flexible way of using ODE as state propagator. From the input files, the robot(s) and object(s) and their properties (such as masses) are read and the ODE bodies are then created using triangular meshes, although in case of simple shapes (such as box, sphere or cylinder), ODE primitive shapes are created. The kinematic tree that represents the robot in the dynamic world is created by adding the joints between the robot bodies. A motor (linear or angular, depending on the joint type) is added to each joint to control the joint velocities and torques. Once the ODE world is created, a dynamic environment class (with the name of the robot, such as *YumiDynamicEnviroment*) is derived from the *OpenDEEnvironment* class provided by OMPL. The derived class reimplementes the functions by defining the control dimensions, control bounds, the way of applying controls, the contact parameters (such as friction, slip, bounce velocity) and the way of evaluating the collisions.

The control dimensions are set equal to the number of actuated degree-of-freedoms and the control bounds define the control (velocity or torque) limits for each joint. A method is provided to define the way of applying the controls. The controls can be joint velocities with maximum allowed torque limits (that the motor can exert to achieve the

desired velocity). Contact parameters are defined between each pair of bodies in contact, describing the interactions. For instance, when an interaction takes place between two bodies, these parameters define how many contact points must be considered, what is the value of the friction coefficient, what is the bounce velocity, what is the constraint force mixing (CFM) and the error reduction parameter (ERP). CFM and ERP are ODE parameters that model the damping and spring behavior of the contact. These contact parameters must be defined carefully because inappropriate values may results in unstable behaviors.

Since physics-based planning allows the dynamic interactions in planning, the way of evaluating collision needs to be modified. Collisions with fixed objects will be forbidden, but collision with movable objects will be allowed, although collision with some movable object may be allowed only from certain parts. For instance, the collision with a car-like object is allowed only from the front or rear side and forbidden along the sides. The differentiation of the objects according to their collision properties is a challenging issue. It is handled by incorporating the contact constraints in the knowledge as explained in Sec. 2.4. The state validity checker will evaluate the satisfaction of the constraints that are imposed by the knowledge.

The state space of each body (robot link or obstacle) in a dynamic environment is 12 dimensional (3 for position, 3 for orientation, 3 for linear velocity and 3 for angular velocity). It is represented as an OMPL *OpenDEStatepace* that is a compound state space with 3 real vector spaces and one SO(3) space for orientation. The state space implements the distance function to measure the distance between two states. The proposed framework provides two implementations of distance function that measure the distance in the workspace and in the configuration space. To measure the former, the position of the TCP is projected in the workspace and the Cartesian distance is measured there. Whereas the latter measures the distance between two configurations.

The implementation of the control space includes different of sampling methods. Since physics-based planning is computationally intensive, the complexity can be reduced by implementing robust control sampling strategies. The current implementation provides a random control sampler, an heuristic-based control sampler and a power-efficient control sampler. The heuristic-based control sampler samples n controls and select the one that results in a state closer to the goal state. The power-efficient control sampler adapts the control sampling strategy according to the region of the state space, i.e. if the robot is in contact with an object the sampling strategy computes the minimum force that is required to push the target object and sample the controls accordingly.

All control-based planners offered by OMPL can be used for knowledge-oriented physics-based planning. For every planner we need to set a pointer to the defined dynamic environment, state space and control space. The planners such as RRT, KPIECE, EST, SyCLoP are already available for planning. Other planners such as SST can be incorporated easily.

2.4 Knowledge Representation and Reasoning

Knowledge is represented with ontologies using OWL, which is a formal way of representing knowledge in term of classes. These classes contain information about the robot (robot kinematic and dynamic properties) and about the environment (the objects

```
%%%%%%%%%%%% READING OBJECT PROPERTIES FROM THE OWL %%%%%%%%%%%%

%Finding the object contact constraint.
find_cont_const(Obj, ContConst):-
    rdfs_individual_of(Obj, sir_mpk:'ManipulatableObject'),
    ( rdf_has(Obj, sir_mpk:'has-contConst', CC),
    rdf_split_url(_, CCSpl, CC), term_to_atom(ContConst, CCSpl);
    \+( rdf_has(Obj, sir_mpk:'has-contConst', _) ), ContConst = null ).
```

(a)

```
%%%%%%%%%%%% READING DATA PROPERTIES FROM THE OWL %%%%%%%%%%%%

find_physical_attributes(Obj, Mass, Friction, GravEff):-
    %Reading the physical attributes.
    rdf_has(Obj, sir_mpk:'has-massValue', M),
    rdf_has(Obj, sir_mpk:'has-frictionValue', F),
    rdf_has(Obj, sir_mpk:'has-gravitationalEffect', G),
```

(b)

```
%%%%%%%%%%%% REASONING ON THE OWL KNOWLEDGE %%%%%%%%%%%%

%Reasoning on object type.
object_classification(Obj, ObjType, Const):-
    find_cont_const(Obj, ContConst), find_manip_const(Obj, ManipConst),
    ( ContConst = null, ManipConst = null, Const = null, ObjType = freely-manipulatable;
    ObjType = constraint-oriented, ( ContConst = null, ManipConst \= null, Const=manip-const;
                                     ContConst \= null, ManipConst = null, Const=cont-const;
                                     ContConst \= null, ManipConst \= null, Const=cont-manip-const ) ), !.
```

(c)

Fig. 4: Examples of Prolog predicates, (a) represents the predicate for the object properties, (b) describes the predicate for the object data properties. (c) describes the predicate for reasoning over the object types.

and their relationship with each other). The relation among classes is defined based on axioms. The axioms are facts that are used for conceptual understanding. We used the protégé editor (http://protege.stanford.edu/) to formulate ontologies (they can be found at https://sir.upc.edu/projects/ontologies/). Domain specific ontologies can be easily defined to handle other planning domain problems, such as task planning.

To enhance the planning process with knowledge, the knowledge is fetched from the ontologies and stored in *instantiated knowledge*. The *instantiated knowledge* is a low-level representation of knowledge that contains the type of the objects, such as manipulatable or fixed, and their contact constraints. These constraints are modeled by specifying regions around the objects, such that the robot can interact with the objects only from these regions. Moreover, other types of constraints can be introduced easily such as the manipulation constraints (constraints over orientation that robot has to maintain during manipulation). The detailed explanation of instantiated knowledge can be found in [18].

The reasoning module is defined in Prolog, which is a language of facts and rules that defines predicates for the knowledge-based reasoning. The predicates are defined in a file (with extension .pl). While creating the ODE world, the Prolog environment is initialized and reads the knowledge from the OWL using predefined predicates. Some examples of the Prolog predicates are shown in Fig. 4. The predicates to access object and data properties are shown in Fig. 4-a and Fig. 4-b respectively. The predicate *find_cont_const(obj, ContConst)* takes the name of ODE body (from ODE world) as input and returns the associated contact constraints. *find_physical_attribute(obj,Mass.Friction,GravEff)* reads the physical properties of the bodies in the ODE world. The predicate described in Fig. 4-c is an example of the reasoning over OWL for the object

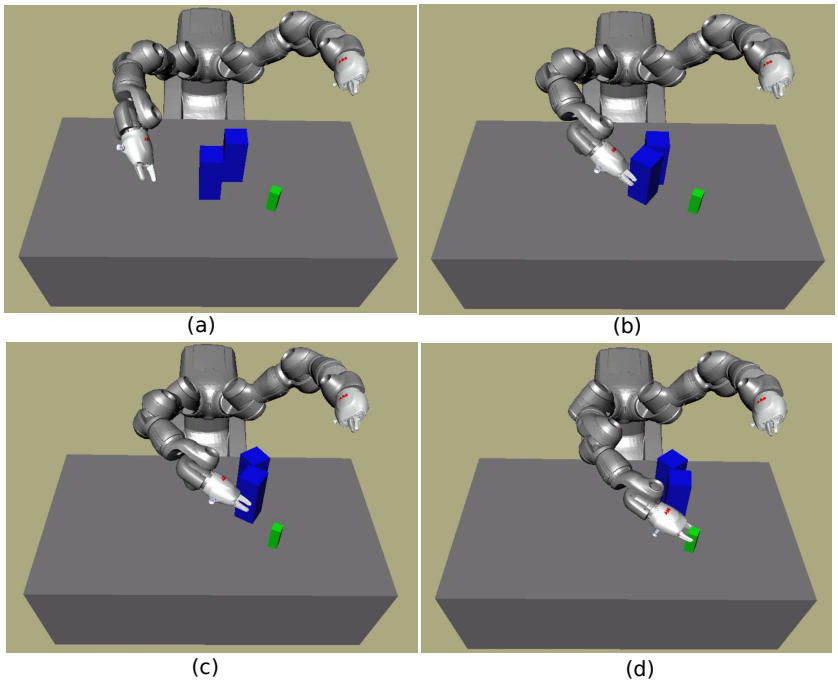

Fig. 5: Visualization of the robot motion with the *Kautham-GUI*

classification. The predicate *object_classification(obj,objectType,Const)* reasons about the types of the object and classify them accordingly into the manipulatable and fixed objects, along with their constraints (contact and manipulation). The Prolog predicates fetch the knowledge from the ontologies and fills the data structures of the *instantiated knowledge* that is used by the motion planner. According to the problem domain, more predicates can be easily defined.

2.5 Visualization

The proposed framework uses the *Kautham-GUI* tool for the visualization of the scene. It provides the visualization of the robot model, the collision model and the visualization of the configuration space (or a projection of it when its dimension is greater than three). The scene can also be visualized with *DrawStuff* (OpenGL based ODE viewer). It provides the visualization of the collision model, the actual robot/object model and the mesh views. Fig. 5 and Fig. 6 depict the visualization using *Kautham-GUI* and *DrawStuff* respectively. The numerical results of a query (such as a list of configurations of the solution path) can also be viewed using *Kautham-Console*, that is a console-based interface of *The Kautham Project*.

2.6 ROS Nodes

The Kautham Project provides a ROS based interface through a node called *Kautham-Node*. It provides the services such as *OpenProblem*, *SetQuery*, *Sovle* and *GetPath*. The current proposal implements two further nodes, *manipulation node* and *communication*

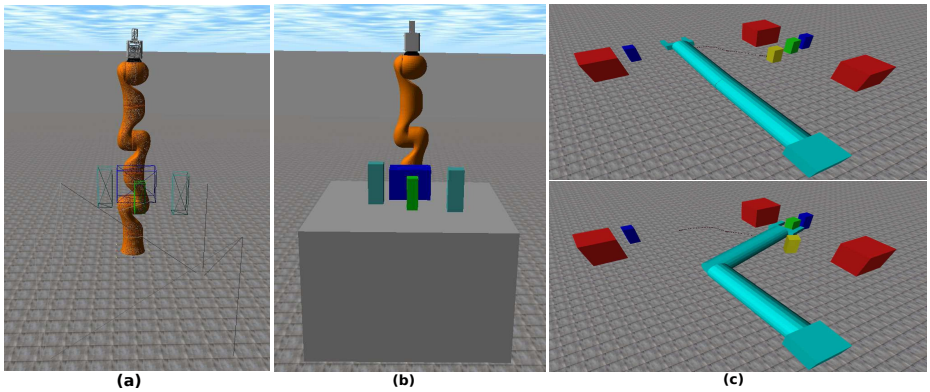

Fig. 6: Example scenes of the visualization of the triangular mesh view and the actual scene view for different robots using *DrawStuff*

manager for handling the manipulation queries and communicating with the real robot or a third party simulation environment, such as Gazebo.

Manipulation Node: The manipulation node extends the functionality of the Kautham node. It is capable of handling the manipulation queries such as push or pull queries. A manipulation query is defined by specifying a target object (that will be pushed or pulled by the robot), the type of manipulation action (such as push, pull or move) and other planning parameters (such as planning time). In response it returns the controls and durations that are to be applied to move the robot from the start to the goal state by satisfying the constraints.

Communication Manager: The communication manager is another ROS node that manages the communication between the software and the real robot. It provides the services to set the query for the manipulation node and sends the computed path to the real robot via ROS/ROS Industrial and receives the feedback from the real robot, such as joint states. Moreover, it also provides the communication between the planning framework and Gazebo or Rviz (http://wiki.ros.org/rviz) to visualize the computed path.

3 Conclusions

This paper described a simulation tool for knowledge-oriented physics-based motion planning by extending *The Kautham Project*. It provides an easy and reliable way to incorporate the rigid- body dynamics and the knowledge-based reasoning (about the action effects) in planning process. It also provides the manipulation node to easily handle the manipulation queries (such as push/pull). The proposed simulation tool also provides an easy way to communicate with real robot through the ROS based communication manager that manages the communication between the proposed tool and the real robot. This simulation tool is used in several research studies, such as [19–21].

References

1. Miller, A.T., Allen, P.K.: Graspit! a versatile simulator for robotic grasping. IEEE Robotics & Automation Magazine **11**(4) (2004) 110–122

2. Suçan, I.A., Chitta, S.: MoveIt! http://moveit.ros.org (2013)
3. Andre, G.: A software architecture for robot control and its application to social robotics, Proc. of the IEEE Int. Conf. on Robotics and Automation: Workshop on Open Source Software in Robotics (2011)
4. Diankov, R.: Automated Construction of Robotic Manipulation Programs. PhD thesis, Carnegie Mellon University, Robotics Institute (August 2010)
5. Vahrenkamp, N., Kröhnert, M., Ulbrich, S., Asfour, T., Metta, G., Dillmann, R., Sandini, G.: Simox: A robotics toolbox for simulation, motion and grasp planning. In: Intelligent Autonomous Systems 12. Springer (2013) 585–594
6. Rosell, J., Pérez, A., Aliakbar, A., muhayyuddin, Palomo, L., García, N., et al.: The kautham project: A teaching and research tool for robot motion planning. In: IEEE Int. Conf. on Emerging Technologies Factory Automation (ETFA). (2014) 1–8
7. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: ROS: an open-source robot operating system. In: ICRA Workshop on Open Source Software. Volume 3. (2009) 5
8. Bruyninckx, H., Soetens, P., Koninckx, B.: The real-time motion control core of the orocos project. In: IEEE Int. Conf. on Robotics and Automation (ICRA),. (2003) 2766–2771
9. Şucan, I.A., Moll, M., Kavraki, L.E.: The Open Motion Planning Library. IEEE Robotics & Automation Magazine **19** (2012) 72–82 `http://ompl.kavrakilab.org`.
10. LaValle, S.M., Kuffner, J.J.: Randomized kinodynamic planning. The Int. Journal of Robotics Research **20**(5) (2001) 378–400
11. Kavraki, L.E., Svestka, P., Latombe, J.C., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE transactions on Robotics and Automation **12**(4) (1996) 566–580
12. Sucan, I., Kavraki, L.E.: A sampling-based tree planner for systems with complex dynamics. IEEE Transactions on Robotics **28**(1) (2012) 116–131
13. Antoniou, G., van Harmelen, F.: Web Ontology Language: OWL. In Staab, S., Studer, R., eds.: Handbook on Ontologies in Information Systems, Springer-Verlag (2003) 67–92
14. Gottschalk, S., Lin, M., Manocha, D., Larsen, E.: Pqp–the proximity query package. http://gamma.cs.unc.edu/SSV/ (1999)
15. Pan, J., Chitta, S., Manocha, D.: Fcl: A general purpose library for collision and proximity queries. In: IEEE Int. Conf. on Robotics and Automation (ICRA), IEEE (2012) 3859–3866
16. Barraquand, J., Langlois, B., Latombe, J.C.: Numerical potential field techniques for robot path planning. IEEE Transactions on Systems, Man, and Cybernetics **22**(2) (1992) 224–241
17. Connolly, C.I., Grupen, R.A.: The applications of harmonic functions to robotics. Journal of Robotic Systems **10**(7) (1993) 931–946
18. Muhayyuddin, Akbari, A., Rosell, J.: Ontological physics-based motion planning for manipulation. In: Proc. of the IEEE Int. Conf. on Emerging Technologies Factory Automation (ETFA). (2015) 1–7
19. Gillani, M., Akbari, A., Rosell, J.: Physics-based motion planning: Evaluation criteria and benchmarking. In: Robot 2015: Second Iberian Robotics Conference, Springer (2016) 43–55
20. Akbari, A., Muhayyuddin, Rosell, J.: Task planning using physics-based heuristics on manipulation actions. In: IEEE 21st Int. Conf. on Emerging Technologies and Factory Automation (ETFA). (2016) 1–8
21. Akbari, A., Muhayyuddin, Rosell, J.: Task and motion planning using physics-based reasoning. In: IEEE 20th Int. Conf. on Emerging Technologies Factory Automation (ETFA). (2015) 1–7