Proceedings of the 2014 IEEE
International Conference on Robotics and Biomimetics
December 5-10, 2014, Bali, Indonesia

Manipulation Task Simulation using ROS and Gazebo

Wei Qian, Zeyang Xia*, Jing Xiong, Yangzhou Gan*, Yangchao Guo,
Shaokui Weng, Hao Deng, Ying Hu, Jianwei Zhang

***Abstract*— This paper intends to create a simulation of manipulator and illustrates the methods of how to implement robot control in a short time. Here we complete the grasp and place mission using Gazebo virtual world and Robot Operating System (ROS). ROS is a distributed framework that is widely used in robotics. Considering the advantages of its easier hardware abstraction and code reuse, ROS was chosen to rapidly organize task architecture and, due to its compatibility with ROS, Gazebo was chosen as the main platform to simulate the designated motion of virtual manipulator.**

I. INTRODUCTION

Simulators have played a critical role in robotics research for quick and efficient testing of new concepts, strategies, and algorithms. A robotics simulator is used to create embedded applications for a robot without depending physically on the actual machine, which saves cost and time. In some case, these applications can be transferred on the real robot without modifications [1].

In the past couple of years, several robot simulators have been developed with different main focus on complexity, accuracy, and flexibility. There are also differences in the possibility of creating and integrating own robot models and virtual environments. Some of the simulators are restricted to a two dimensional environment or are only approximating dynamics and realistic interaction of the robots with the environment.

Robot Operating System (ROS) [2] is packaged libraries and tools to help create robot applications. There are numerous contributions from around the world. Gazebo [3] is the 3D simulation that is part of the Player [4] Project. Gazebo is designed to accurately reproduce the dynamic environments a robot may encounter. There are some famous robots simulated in ROS and Gazebo platform, such as PR2, Care-O-bot, TurtleBot, etc. In this case, we can draw a conclusion that Gazebo simulator based on ROS is a powerful tool for robot simulation. However, there are some difficulties to make the simulator do what we want, especially for beginners. This paper is aimed to present the idea that how to easily understand and make use of ROS and Gazebo.

Robot Manipulator simulation has been relatively matured over the past two decades, however in order to model the state of the art in robotics with new actuators and advanced control algorithms as well as realistic physics, there is a need to have improved simulation tools as evident by the Gazebo project to develop a tool for DARPA [5]. On the other hand, open source models and sharing the code is one of the most important solutions to increase the interaction among robotic research groups and to provide a common tool for testing various algorithms on a single environment. OpenHRP [6], the recent Gazebo and ROS projects are two of the latest projects following this approach.

This paper is organized as follows: Section 2 describes the general architecture of ROS and Gazebo, Section 3 shows how to create a basic manipulator model to simulate. Section 4 depicts the operation in Gazebo and the implementation of a simulating experiment. The paper ends with the conclusion in Section 5.

II. ARCHITECTURE

The philosophy of ROS is to make a piece of software that could work in other robots by making little changes in the code [7]. There are some significant terms specified in ROS we have reasons to know first. ROS is based on nodes, messages, topics, and services.

The ROS Master is the core of ROS since it provides name registration and lookup to the rest of the Computation Graph. You can take it as an on-off switch of electrical equipment. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services. In ROS, all major functionality is broken up into a number of chunks that communicate with each other using messages. Each chunk is called a node and typically runs as a separate process. Nodes communicate with each other by passing messages. A message is simply a data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types.A node sends out a message by publishing it to a given topic. The topic is a name that is used to identify the content of the message. A node that is interested in a certain kind of data will subscribe to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish or subscribe to multiple topics. Request/reply is done via services, which are defined by a pair of message structures: one for the request and the other for the reply. A providing node offers a service under a name and a client uses the service by sending the request message and awaiting the

This research was supported by National Science Foundation of China (No. 51305436), Shenzhen High-level Oversea Talent Program (Peacock Plan) (No. KQCX20130628112914284).

W. Qian is with Harbin Institute of Technology Shenzhen Graduate School, and Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, and The Chinese University of Hong Kong. Z. Xia, Y. Gan, Y. Guo, S. Weng, H. Deng and Y. Hu are with Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, and The Chinese University of Hong Kong, China. J. Xiong is with Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. J. Zhang is with TAMS, Department of Informatics, University of Hamburg, Germany.

*Author to whom correspondence should be addressed. Email: {zy.xia, yz.gan}@siat.ac.cn; Phone: +86-755-8639 2181

978-1-4799-7397-2/14/$31.00 ©2014 IEEE

reply. ROS client libraries generally present this interaction to the programmer as if it were a remote procedure call.

In addition, ROS provides enough tools for data process and analysis like 3D visualization (RViz), logging real-time robot experiments and playing them offline with (rosbag/rxbag), plotting data (rxplot) and visualizing the entire ROS network structure (rxgraph).

Gazebo simulator can be divided into libraries for physics simulation, rendering, user interface, communication, and sensor generation. Gazebo can simulate robot in a three-dimensional world. It generates both realistic sensor feedback and physically plausible interactions between objects (it includes an accurate simulation of rigid-body physics).

Although Gazebo has been an independent system that stands alone outside of ROS in the latest ROS version, we can also make it work as a node that simulates robot motion and exchanges data with other nodes.

III. MANIPULATOR MODELING

The Unified Robot Description Format (URDF) [8] is an XML specification to describe a robot. URDF files is used to record the whole information of a virtual robot. It's common to create robot model by Computer Aided Design (CAD) tools such as Solidworks, Pro-engineer, Blender, etc. However, there must exist a transformation from CAD model to URDF robot description model.

The basic structure of a robot is always divided into links and joints no matter how complex the robot is (i.e. Fig. 1).

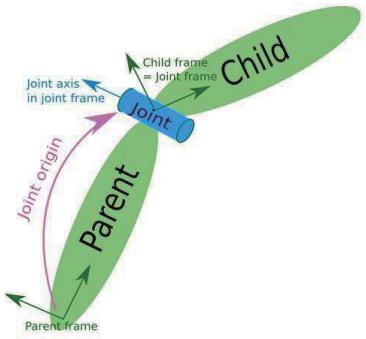

Fig. 1. Connection of two links by one revolute joint. Every link has its frame which cannot be transformed after exported from CAD model. The joint frame can be transformed from parent frame, and child frame can also be transformed from joint frame. The key point is to make all frames have a correct spatial relationship.

In URDF robot description model,The whole elements between tag <link> and tag </link> are the description of one specified link, the same as joint description between tag <joint> and tag </joint>.

STereoLithography (STL) [9] and Collada [10] files can be imported as meshes into URDF. The recommended format for best texture and color support is Collada files, though STL files are also supported.

What we need to do is to make scattered links together. Commonly, a joint is the connection of two different links. And the two links can be divided into parent link and child link. Since every link has its fixed own coordinate frame. The transformation information from parent link to child link must be filled into joint property block.

RViz is a extremely useful visualization tool of ROS which can be used to display the topics communicated between nodes. RViz has a Graphical User Interface (GUI) to allow users to configure and modify robots.

Launching RViz node and other essential nodes to establish a ROS network which can control the position of each joint provides a simple method to check the joint connection between links.

Here are some brief introduction about main active nodes and topics transported between each other(i.e. Fig. 2).

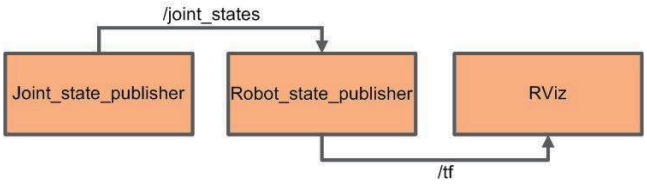

Fig. 2. When we launch RViz, joint_state_publisher, robot_state_publisher and RViz nodes(orange rectangle frames), topics named joint_states and tf are transported according to the arrow directions.

joint_state_publisher: This node reads the robot_description parameter and publishes all non-fixed joints' information such as position, velocity, acceleration, at a specified frequency. It also provides a control GUI to set the joint position of manipulators.

robot_state_publisher: This node subscribes to current joint positions of the manipulators and publishes transform data of coordinate frames attached to each link by the method called forward kinematics.

tf: The tf library was designed to provide a standard way to keep track of coordinate frames and transform data within an entire system such that individual component users can be confident that the data is in the coordinate frame that they want without requiring knowledge of all the coordinate frames in the system [11] .

IV. GAZEBO SIMULATION

Gazebo is a multi-robot simulator for outdoor environments. Like Stage [12] (part of the Player project), it is capable of simulating a population of robots, sensors and objects, but does so in a three-dimensional world. It generates both realistic sensor feedback and physically plausible interactions between objects (it includes an accurate simulation of rigid-body physics).

By realistically simulating robots and environments code designed to operate a physical robot can be executed on an artificial version. Numerous researchers have also used

Gazebo to develop and run experiments solely in a simulated environment. Controlled experimental setups can easily be created in which subjects can interact with manipulators in a realistic manner.

There is a big difference between RViz visualisation and Gazebo simulation, the former one is used to display relative position of links, but the later one can be regarded as a experimental copy of real robot in virtual world.

A transmission is an element in a control pipeline that transforms efforts/flow variables. Transmission-specific code (not robot-specific) implementing bidirectional effort and flow maps under a uniform interface shared across transmission types. Gazebo controllers need the corresponding transmission interface in URDF file. We can configure the effort controller transmission of a joint named joint1 rapidly by the block of code below.

```
<transmission name="tran1">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="joint1"/>
  <actuator name="motor1">
    <hardwareInterface>EffortJointInterface</
        hardwareInterface>
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>
```

In addition to the transmission tags, a Gazebo plugin needs to be added to our URDF that actually parses the transmission tags and loads the appropriate hardware interfaces and controller manager. The gazebo_ros_control plugin is very simple, though it is also extensible via additional plugin architecture to create custom robot hardware interfaces between ros_control and Gazebo. Adding the block of code below to active the gazebo_ros_control plugin to allow us to start a list of controller_manager services, which can be used to list, start, stop or switch controllers.

```
<gazebo>
  <plugin name="gazebo_ros_control" filename="
      libgazebo_ros_control.so">
    <robotNamespace>/MYROBOT</robotNamespace>
    <robotSimType>gazebo_ros_control/DefaultRobotHWSim</
        robotSimType>
  </plugin>
</gazebo>
```

In order to let our manipulator work as we want, robot controller is an essential part. Trajectory controller is a kind of position tracking controller, it quintic algebra curves Making use of the trajectory controller can be used as a controller easily, we should load a YAML [13] configuration file that corresponds with the joint trajectory controller. In the configuration file, we set the controller types, joint names and PID parameters.

```
arm_controller:
  type: "effort_controllers/JointTrajectoryController"
  joints:
    - joint1
  constraints:
    goal_time: 5.0
    joint1:
      goal:       0.05
      trajectory: 0.05
```

The code above set the controller's type, joint name and some constrains, such as

When the joint_trajectory_controller is launched and running, there are two ways can be used to publish commands to control the joints (i.e. Fig. 3).

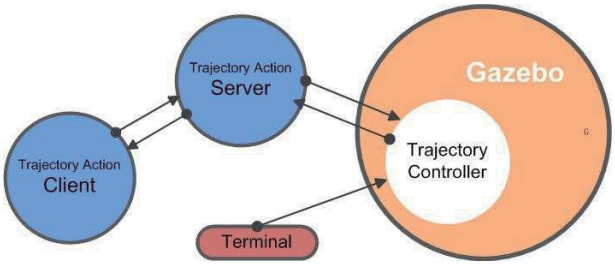

Fig. 3. There are two methods to command the trajectory controller inside Gazebo simulator. Directly command message from the terminal is a simple way to control our robot. The other method is to use action client to communicate with action server so as to command the controller.

The first method is simply open a terminal to send a specified-format topic to the joints. In this way, we can do some simple tasks, for example, send a simple instruction to home all the joints to initial position. The command below is intended to initialize my 7DOF virtual mechanical arm in three seconds.

```
$ rostopic pub /arm_controller/command trajectory_msgs/
    JointTrajectory '{joint_names:['joint1', 'joint2', '
    joint3', 'joint4', 'joint5', 'joint6', 'joint7'],
    points: [{positions: [0,0,0,0,0,0,0], time_from_start
    : {secs: 3}}]}'
```

The other method is to create a specified action client to communicate with action server which has been capsuled into trajectory controllers (i.e. Fig. 4).

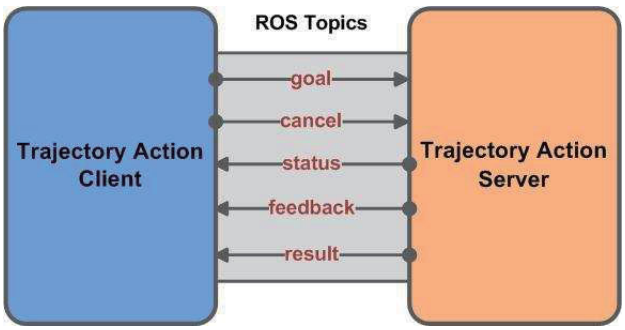

Fig. 4. Topics Communicated between action client and action server. This mechanism allows task preemption and process feedback.

The actionlib[15] package provides tools to create servers that execute long-running goals which can be preempted. It also provides a action client in order to send requests to the server.

We need to define action messages to connect action client and server. An action specification defines the Goal, Feedback, and Result messages with which clients and servers communicate:

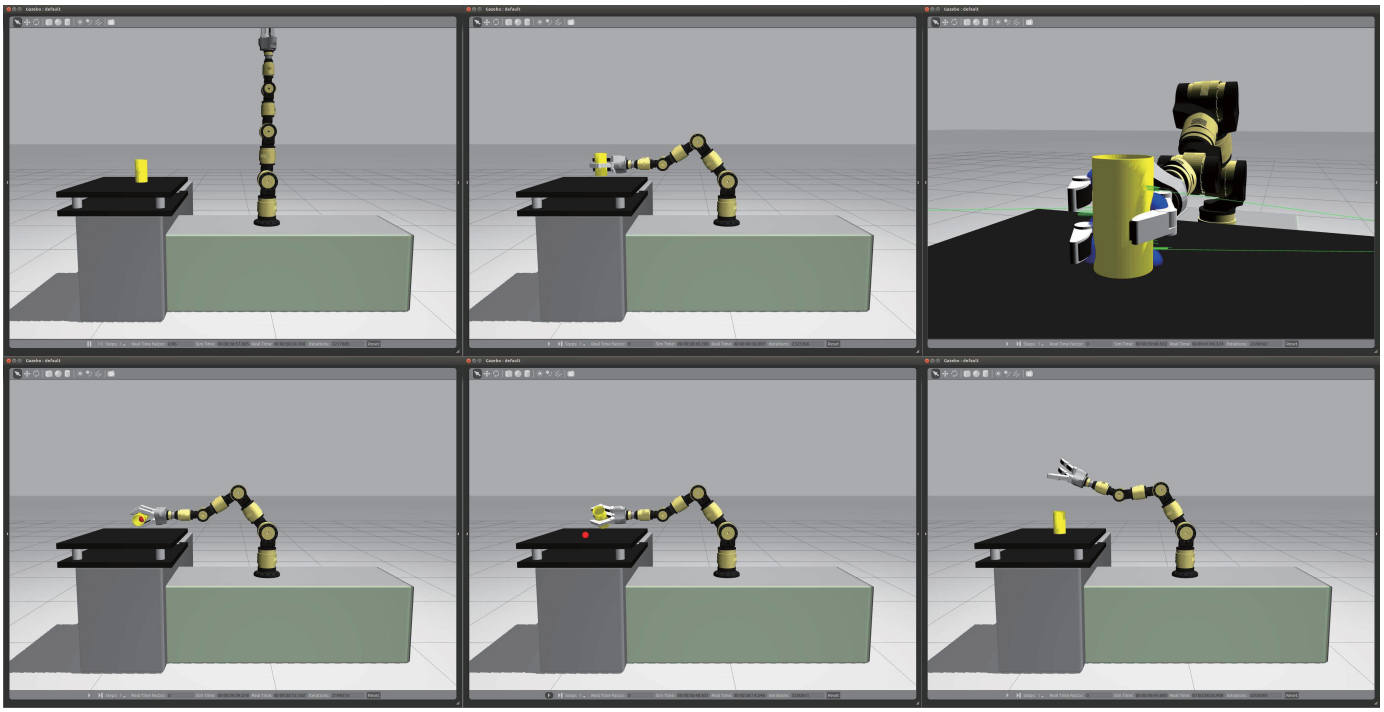

Fig. 5. The whole process of the task pouring out a red ball from the pale green cup. First figure shows the beginning preparation and scenario. The next shows the robot arm moved to the pre-grasp position, then the three fingers dexterous hand close and lift the cup up to defined height to pour the small ball. Manipulator rotates the end link of arm to make the cup overturn. We can see that the red ball was poured out from the blue cup. Completing this task only need no more than one minute that a little longer than operating by human hand.

Goal: To accomplish tasks using actions, we introduce the notion of a goal that can be sent to an Action Server by an Action Client.

Feedback: Feedback provides server implementers a way to tell an Action Client about the incremental progress of a goal.

Result: A result is sent from the Action Server to the Action Client upon completion of the goal. This is different than feedback, since it is sent exactly once.

By Making use of the actionlib package, we can establish specified ROS network to implement simple task easily. Trajectory action client send specified goal to action server, commonly called FollowJointTrajectoryGoal. If we have knew the waypoints' postions, velocities ,accelerations, we can create a goal containing all the information about final robot state. The function called toMsg below is used to convert desired joint position to specified goal for action client to send to Server(Here we create several static waypoints since their velocities are equal to zero).

```
//Function used to convert desired joint position to
     specified goal.
control_msgs::FollowJointTrajectoryGoal toMsg(const Eigen
     ::VectorXd& joint)
{
    control_msgs::FollowJointTrajectoryGoal goal;

    goal.trajectory.joint_names.push_back("joint1");
    goal.trajectory.joint_names.push_back("joint2");
    goal.trajectory.joint_names.push_back("joint3");
    goal.trajectory.joint_names.push_back("joint4");
    goal.trajectory.joint_names.push_back("joint5");
    goal.trajectory.joint_names.push_back("joint6");
    goal.trajectory.joint_names.push_back("joint7");

    goal.trajectory.points.resize(1);
    goal.trajectory.points[0].positions.resize(7);

        for(unsigned int i=0; i<7; ++i)
        goal.trajectory.points[0].positions[i] = joint(i);

    return goal;
}
```

We should create an action client to connect with a specified name action server. The desired joint position can be got from inverse kinematics, and then convert it into the form of ROS goal msg. After that, action client sends goal to server and waits for the result of the whole sub process. Part of the C++ code is displayed below.

```
typedef actionlib::SimpleActionClient<control_msgs::
    FollowJointTrajectoryAction> TrajClient;
//Creating a action client to communicate with action
    server named "/arm_controller/follow_joint_trajectory
    ".
traj_client = new TrajClient("/arm_controller/
    follow_joint_trajectory", true);

while(!traj_client->waitForServer(ros::Duration(5.0)))
{
    ROS_INFO("Waiting for server");
}

control_msgs::FollowJointTrajectoryGoal goal;
//Using Eigen linear algebra C++ template library.
Eigen::VectorXd joint(7),
joint<<0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0;
//Converting joint position to goal type
goal = toMsg(goal);
goal.trajectory.header.stamp = ros::Time::now();

traj_client->sendGoal(goal);
```

By recycling the sub process until completing task planning can simulate many common tasks. The structure diagram below(i.e. Fig. 6) can be regarded as a basic method to make our virtual robot do what we want.

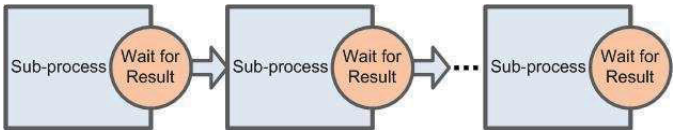

Fig. 6. Topics Communicated between action client and action server. This mechanism allows task preemption and process feedback.

Here we want to use our virtual 7 DOF manipulator to pour out a small ball from the cup on the desk nearby(i.e. Fig. 5). This grasp and place task can be divided into five parts, (a)move to pre-grasp position, (b)grasp the cup, (c)pour out the object inside the cup, (d)place the cup, (e)move to initial position. By setting a few trajectory waypoints which dividing the whole process into several separate parts. Using the method above, we merely need to do is computing the grasp position and orientation and adding some blocks of code. After compiling and debugging, our node worked as a trajectory action client can communicate with trajectory action server. The whole process is very steady and continuous and we think the the second method can be taken as the first choice to do complex task simulation.

CONCLUSION AND FUTURE WORKS

Gazebo simulator becomes a powerful tool when it's worked as a node in ROS environments. It is a relatively simple way to simulate our robot to complete ordinary tasks in daily life. As a result, the overall time spent in robotics research is greatly reduced due to code reuse. Sometime, the process of designing and building the hardware alone would have consumed numerous months. However, Gazebo reduced the time frame down to less than two months, during which time both the physical structure and software were modified and tested in parallel. Clearly, using of this feature allows developers to easily move from basic concepts to real working systems in a short period of time.

Since part of our work is based on the preliminary understanding of ROS and Gazebo, and what we want to do is showing the methods of how to simulate our virtual manipulator in a more effective way.

Further research is needed on the simulation of algorithm, one of the main work will be motion planning algorithm simulation in cluttered environment. In addition, there are many virtual sensors in Gazebo, such tactile sensor, force/torque sensor, Kinect, etc, which can make manipulator grasp and place more steadily, this is also part of our future research contents.

REFERENCES

[1] C. Pizarro, Patricio, T.V. Arredondo, and M.T. Torriti, "Introductory Survey to Open-Source Mobile Robot Simulation Software." *Robotics Symposium and Intelligent Robotic Meeting (LARS), 2010 Latin American.* IEEE, 2010.

[2] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler and A. Ng, "ROS: an open-source Robot Operating System," *ICRA Workshop on Open Source Software*, 2009.

[3] N. Koenig and A. Howard, "Design and Use Paradigms for Gazebo, An Open-source Multi-Robot Simulator," in *International Conference on Intelligent Robots and Systems*, Sendal, Japan, 2004.

[4] Online, 2013, IEEE Spectrum: DARPA awards simulation software contract to open source robotics foundation, URL: http://spectrum.ieee.org/automaton/robotics/robotics-software/darpa-robotics-challenge-simulation-software-open-source-robotics-foundation

[5] B. P. Gerkey, R. T. Vaughan and A. Howard, "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems," *Proceedings of the International Conference on Advanced Robotics (ICAR 2003)*, pp. 317-323, 2003.

[6] Online, 2013, OpenHRP Dynamic Simulator, URL: http://www.openrtp.jp/openhrp3/en/index.html

[7] W. Garage, ROS: Robot Operating System, 2011[J]. URL: http://www.ros. org/(ultimo acesso: 10/12/2011), 2011.

[8] L. Kunze , T. Roehm , M. Beetz, "Towards semantic robot description languages." Robotics and Automation (ICRA), 2011 IEEE International Conference on. IEEE, 2011.

[9] Lee, K. H., H. Woo, and T. Suk, "Data reduction methods for reverse engineering." *The International Journal of Advanced Manufacturing Technology* 17.10 (2001): 735-743.

[10] Miyahara, Katsunori, and Y. Okada, "COLLADA-based File Format Supporting Various Attributes of Realistic Objects for VR Applications." *Complex, Intelligent and Software Intensive Systems, 2009. CISIS'09. International Conference on*. IEEE, 2009.76.

[11] T. Foote, "tf: The transform library." Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on. IEEE, 2013.

[12] R. Vaughan, "Massively multi-robot simulation in stage." Swarm Intelligence 2.2-4 (2008): 189-208.

[13] V. Sinha, F. Doucet, C. Siska, R. Gupta, S. Liao, and A. Ghosh, "YAML: a tool for hardware design visualization and capture." *Proceedings of the 13th international symposium on System synthesis. IEEE Computer Society*, 2000.

[14] R. Miller, "Configuration management with Subversion, YAML and Perl template toolkit." *Proceedings of the 5th International System Administration and Network Engineering Conference SANE*. Vol. 6. 2006.

[15] J. Bohren, S. Cousins, "The SMACH high-level executive [ROS news]." *Robotics & Automation Magazine, IEEE* 17.4 (2010): 18-20.