Enabling codesharing in Rescue Simulation with USARSim/ROS

Kootbally, Z.; Balakirsky, S.; Visser, A.

DOI
10.1007/978-3-662-44468-9_54

Publication date
2014

Document Version
Author accepted manuscript

Published in
RoboCup 2013: Robot World Cup XVII

Citation for published version (APA):
Kootbally, Z., Balakirsky, S., & Visser, A. (2014). Enabling codesharing in Rescue Simulation with USARSim/ROS. In S. Behnke, M. Veloso, A. Visser, & R. Xiong (Eds.), RoboCup 2013: Robot World Cup XVII (pp. 592-599). (Lecture Notes in Computer Science; Vol. 8371), (Lecture Notes in Artificial Intelligence). Springer. https://doi.org/10.1007/978-3-662-44468-9_54
Enabling codesharing in Rescue Simulation with USARSim/ROS

Zeid Kootbally¹, Stephen Balakirsky², and Arnoud Visser³

¹ Department of Mechanical Engineering, University of Maryland
² Georgia Tech Research Institute, Atlanta, Georgia
³ Intelligent Systems Laboratory Amsterdam, Universiteit van Amsterdam

Abstract. The Robot Operating System (ROS) has been steadily gaining popularity among robotics researchers as an open source framework for robot control. The Unified System for Automation and Robot Simulation (USARSim) has been used for many years by robotics researchers and developers as a validated framework for simulation. This paper presents a new ROS node that is designed to seamlessly interface between ROS and USARSim. It provides for automatic configuration of ROS transforms and topics to allow for full utilization of the simulated hardware. The design of the node as well as examples of its use for mobile robot inside the RoboCup Rescue Simulation League are presented.

1 Introduction

With the development of advanced but also more complex algorithms one cannot expect that a robotic control system will be developed from scratch. With the aid of open source projects such as the Robot Operating System (ROS) [17] allow anyone with a Linux computer to download and run some of the most advanced robotic algorithms that exist. This is essential for the RoboCup mission; to accelerate the developments of intelligent and dexterous robots. With working modules that cover all basic capabilities needed for a functional robot, developers can concentrate on improving the aspects needed for their application.

Most programmers have access to a single robot or small sensor suite, but are missing access to some of the robotic hardware needed for the job. Simulators exist to fill this void and allow both experts and novices to experiment with robotic algorithms in a safe, low-cost environment. However, to truly provide valid simulation, the simulator must provide noise models for sensors and must be validated [2, 13-15, 21]. One modern robotic simulator, known as the Unified System for Automation and Robot Simulation (USARSim) [2] provides such a simulation platform. This simulator has been used by the expert robotics community for several years and has played an important role in developing robotics applications [3].

This paper examines how a new interface to the ROS control framework (introduced by [5]) can be used inside the RoboCup Rescue Simulation League [1]. The ROS framework allows for easy sharing of modules [9], allowing fast progress.
BACKGROUND

The USARSim Framework

USARSim [7] is a high-fidelity physics-based simulation system based on the Unreal Developers Kit (UDK)\(^4\) from Epic Games. Through its usage of UDK, USARSim utilizes the PhysX physics engine [6] and high-quality 3D rendering facilities to create a realistic simulation environment that provides the embodiment of, and environment for a robotic system. The current release of USARSim consists of various environmental models, models of commercial and experimental robots, and sensor models. High fidelity at low cost is made possible by building the simulation on top of a game engine. By loading the most difficult aspects of simulation to a high volume commercial platform (available for free to academic users) which provides superior visual rendering and physical modeling, full user effort can be devoted to the robotics-specific tasks of modeling platforms, control systems, sensors, interface tools and environments. These tasks are in turn accelerated by the advanced editing and development tools integrated with the game engine. This leads to a virtuous spiral in which a wide range of platforms can be modeled with greater fidelity in a short period of time.

![Fig. 1. Sample of 3D environments in USARSim.](image)

USARSim was originally based upon simulated environments in the USAR domain [20]. Realistic disaster scenarios as well as robot test methods were cre-

\(^4\) http://www.unrealengine.com/udk
USARSim has been used worldwide and more environments have been developed for different purposes, including human robot interaction and educational games [3]. Other environments such as the NIST campus (Figure 1(c)) and factories (Figure 1(d)) have been used to test the performance of robotic algorithms in different circumstances [2, 21]. The simulation is also widely used for the RoboCup Virtual Robot Rescue Competition [1], the IEEE Virtual Manufacturing and Automation Challenge [4].

The ROS Framework

ROS [17] is an open source framework designed to provide an abstraction layer to complex robotic hardware and software configurations. It provides libraries and tools to help software developers create robot applications and has found wide use in both industry and academia [8]. Examples of ROS applications include Willow Garage's Personal Robots Program [22], the Stanford University STAIR project [16] and the European Nifti project [11]. Developers of ROS code are encouraged to contribute their code back to the community and to provide documentation and maintenance of their algorithms.

ROS consists of a large range of tools and services that both users and developers alike can benefit from. The philosophical goals of ROS include an advanced set of criteria and can be summarized as: peer-to-peer, tools-based, multi-lingual, thin, and free and open-source [17]. Furthermore, debugging at all levels of the software is made possible with the full source code of ROS being publicly available. Thus, the main developers of a project can benefit from the community and vice-versa.

THE ROS/USARSIM INTERFACE

USARSim is designed to communicate over a TCP/IP socket with a computer hosting the controller of the robot. The robot is “spawned” into the simulated world running on the game server. A robot’s configuration is controlled by an initialization file that resides on the simulation system’s computer, comparable with the launch file from Gazebo. This file controls aspects such as sensor configuration, battery life, and simulated noise models. One socket connection is established per simulated robot, with both commands and sensor data being transmitted over the socket. An additional separate socket is established for high-volume sensors such as camera systems.

ROS stacks are designed to “bottom out” at a hardware abstraction layer that provides the interface to sensors and the motors of the robot; publishing and subscribing to the basic topics of the robot. For example, the mobility stack expects to be able to control a platform by writing commands to low-level topics that control items such as the velocity. In addition, the mobility stack expects feedback from sensors, such as the movement detected by the inertia sensor. In order to close this low-level loop between ROS and USARSim, a USARSim
The **RosSim** node relies on several parameters for its configuration. These are necessary for the creation of a robot in USARSim and a transform tree **tf** in ROS. A transform tree for the P3AT robot is shown in Figure 2. This transform tree is built automatically from data obtained from the USARSim GEO and CONF messages. Since USARSim supports multiple sensors on a robot, it should be specified which sensor should be preferred for an initial localization estimation (later updated in a SLAM algorithm). That sensor’s name is automatically changed to **odom**. The **base_footprint**, representing the robot platform and the **base_link** representing robot sensor mounting points are also automatically generated.

Vehicle movement commands into USARSim vary depending on the robot type. For example, skid-steered vehicles require left and right wheel velocities while Ackerman steered vehicles require steering angle and linear velocity. ROS provides a **cmd_vel** topic that includes both linear and angular velocities. The **RosSim** node automatically converts these velocities into the appropriate commands and values for the USARSim simulator based on the robots steering type.

---

5 http://sourceforge.net/projects/usarsimros/
wheelbase, and wheel separation. Vehicle speeds are also clamped to not exceed maximum velocities that are set in the simulation.

**Sensor Interface**

ROS provides a rich vocabulary of sensor interface messages. The RosSim node strives to automatically match simulated sensors to the appropriate ROS topic. Currently, a wide variety of sensors are supported: ground truth, inertial navigation, GPS, range-scanners, sonar, CO2 and acoustic sensors are supported. It is our ambition to support all sensors which are used in the RoboCup Rescue competition [1]. This presence of sensors is queried via USARSim’s CONF and GEO messages and automatically included the robot transform tree used by ROS. The corresponding sensor messages are published at the rate that the RosSim node receives the sensor output.

**Mobile Robot Control with the ROS Navigation Stack**

Control of mobile robots through the ROS/USARSim interface is performed with the ROS navigation stack\(^6\). The navigation stack provides for 2D navigation and takes in information from odometry, sensor streams, and a goal pose to plan the velocity commands that does not lead to collisions. The planning for collision free paths is performed on two levels: one on a local costmap and a global costmap.

Although different models of mobile robot are developed in USARSim, the Kenaf has proven its worth in the Rescue League [14]. The Kenaf robot (see Figure 3) is a challenge to be used the navigation stack with its elongated rectangular form (so not square or circular) and its flippers. As configured in our experiments, it includes a laser scanner mounted on its base.

**Low-level Navigation** The ROS/USARSim interface allows for the start-up and control of the default robot base controllers by directly sending velocity commands to the base. This task was performed using the following commands:

\(^6\) {http://www.ros.org/wiki/navigation}
1. Bring up an environment in USARSim.
2. `$roscore`
3. `$roslaunch usarsim usarsim.launch`
4. `$rosrun teleop_twist_keyboard teleop_twist_keyboard.py`
5. `$rosrun gmapping slam_gmapping scan:=lms200 _odom_frame:=odom`

In step 1, an environment is started on the server side (USARSim). If an environment is not up and running, passing messages between ROS and USARSim will fail. Step 2. starts roscore, a collection of nodes and programs that are pre-requisites of a ROS-based system for ROS nodes to communicate. Step 3. launches the usarsim.launch file. This launch file contains the parameters to connect the game server and starts the RosSim node that provides a connection between ROS and USARSim. Step 4. starts the teleop_twist_keyboard node which sends velocity commands to the RosSim node through the computer keyboard. At this point, the Kenaf can be controlled by keyboard teleop in the USARSim environment. Step 5. starts the node slam_gmapping which transforms each incoming scan from the laser into the odometry tf frame to build a map. Here, the topic `scan` is used to create the map with the parameter `_odom_frame`, the frame attached to the odometry system.

![Fig 4. Mobile robot control using teleop](image)

Figure 4 is a graph generated by `rxgraph` with the option “quiet”. The graph illustrates the communication between the nodes RosSim, teleop_twist_keyboard, and slam_gmapping. The keyboard inputs are converted in velocity commands and then communicated to the RosSim node on the topic `cmd_vel`. slam_gmapping uses the topics (`lms200`) and (`tf`) as inputs to build the map. To save the generated map, the following command is used:

```bash
$rosrun map_server map_saver
```

Figure 5(a) is a bird's eye view of the environment used to run the teleop command to steer a robot through the environment and Figure 5(b) is the map generated by the map_saver utility-command.

**High-level Navigation** Several teams [18,19] are currently building high-level navigation software by combining existing ROS algorithms with algorithms from
their institutes. Those teams perform simultaneous localization and mapping to generate online maps, and use those maps to plan path (for instance with the RRT algorithm [19]) to replace *teleop* with autonomous navigation.

**CONCLUSION AND FUTURE WORK**

This paper has presented a new ROS package that allows for the seamless interface of USARSim with ROS. The package provides for auto-discovery of robots and sensors, and produces the standard ROS topics that one would expect from a physical platform. Several researchers from outside the RoboCup community [10, 12] have already tried this interface. Still, further work is needed to provide the control interface for legged robots (as the Nao robot) and flying robots (such as the AirRobot and the AR.Drone). In addition, the whole sensor suite must have their USARSim interfaces wrapped to be supported in the ROS environment. On the positive side, as the Darpa Robotics Challenge is based on the ROS framework, progress in this competition could be an advance for the RoboCup, and vice versa.

**References**


