Proceedings of the ASME 2012 International Symposium On Flexible Automation
ASME/ISCIE 2012
June 18-20, 2012, St. Louis, Missouri, USA

DETC2010/MECH-12345

TITLE GOES HERE

Stephen Balakirsky*
Intelligent Systems Division
National Institute of Standards and Technology
Gaithersburg, Maryland 20899
Email: stephen.balakirsky@nist.gov

Zeid Kootbally
Department of Mechanical Engineering
University of Maryland
College Park, Maryland, 20742
Email: zeid.kootbally@nist.gov

ABSTRACT

Abstract goes here (less than 150 words).

INTRODUCTION

Robots have now become a part of many people's everyday lives. Whether as simple toys used by children, floor cleaning robots used in the home, or high precision industrial manipulators used in manufacturing, these systems are quickly changing the ways in which we play and work. One can no longer make the assumption that robots will exist in enclosed areas, or that the programmer or developer of the systems will be highly-skilled robotics experts. Indeed, new open source projects in robotic control systems such as the Robot Operating System (ROS)[1] [1] allow anyone with a Linux computer to download and run some of the most advanced robotic algorithms that exist. If the users desire a deeper knowledge of how these algorithms work, there is even a free robotics course from Stanford that may be taken online.

One thing that many of these individuals are missing is robotic hardware. 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. One modern robotic simmulator, know as the Unified System for Automation and Robot Simulation (USARSim) [2] provides such a simulation platform. This simmulator has been used by the expert robotics community for several years and has played an important role in developing robotics applications. Its uses includehave included rapid prototyping, debugging, and development of many tasks ranging from legged robots playing soccer [3] to urban search and rescue [4, 5]. In fact, a search for the keyword "USARSim" on Google Scholar returns over 700 articles that have referenced the simulation platform.

One reason for the simulation environment's popularity is that it enables researchers to focus on algorithm development without having to worry about the hardware aspects of the robots. Simulation can be an effective first step in the development and deployment of new algorithms and provides extensive testing opportunities without the risk of harming personnel or equipment. Major components of the robotic architecture (for example, advanced sensors) can be simulated and enable the developers to focus on the algorithms or components in which they are interested without the need to purchase expensive hardware. This can be useful when development teams are working in parallel or when experimenting with novel technological components that may not be fully implemented or available.

Simulation can also be used to provide access to environments that would normally not be available to the development team. Particular test scenarios can be run repeatedly, with the assurance that conditions are identical for each run. The environmental conditions, such as time of day, lighting, or weather, as well as the position and behavior of other entities in the world can be fully controlled. In terms of performance evaluation, it

*Address all correspondence to this author.

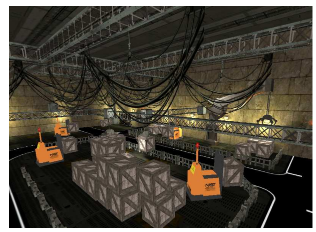
[1]Certain commercial software and tools are identified in this paper in order to explain our research. Such identification does not imply recommendation or endorsement by the authors, nor does it imply that the software tools identified are necessarily the best available for the purpose.

can truly provide an “apples-to-apples” comparison of different software running on identical hardware platforms in identical environments. Another important feature of a robotic simulator is easy integration of different robotic platforms, different scenarios, different objects in the scene, as well as support for multi-robot applications.

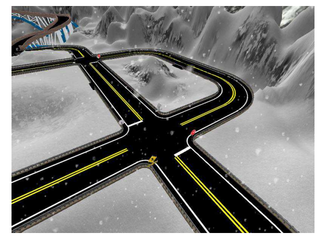
This paper examines a new interface that allows the ROS control framework to communicate directly with USARSim thus opening up sophisticated robot control and development to an entirely new audience. Novice robot developers can now work with world class algoirthms from the safety of their computer without the expense of actual robotic hardware. This paper describes the interface connecting the USARSim framework with the ROS framework. The following sections describe, analyze and illustrate the new interface for the navigation of a mobile robot base, control of a robotic arm, and interface to existing sensors. In addition, a novel sensor interface is presented that allows the simulator to mimic a sensor processing system that produces the 6-degree-of-freedom pose for known objects.

(a) Test Room,

(b) NIST main campus.



(c) Factory,



(d) ARDA.

FIGURE 1. Sample of 3D environments in USARSim.

BACKGROUND

In order to experiment with robotic systems, a researcher requires a controlable robotic platform, a control system that interfaces to the robotic system and provides behaviors for the robot to carry out, and an environment to operate in. This paper examines an opensource (the game engine is free, but license restrictions do apply), freely available framework capible of fulfilling all of these requirements. This framework is composed of the USARSim framwork that provides the robotic platform and environment, and the ROS framework that provides the control system.

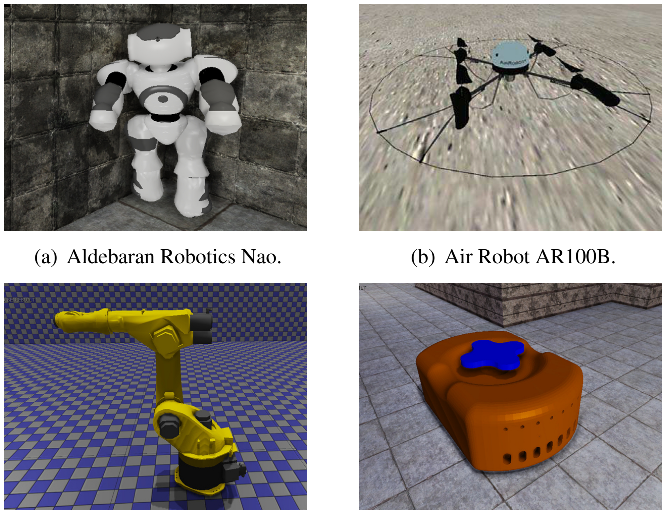
The USARSim Framework

USARSim [4, 5] is a high-fidelity physics-based simulation system based on the Unreal Developers Kit (UDK) [6] from Epic Games. USARSim was originally developed under a National Science Foundation grant to study Robot, Agent, Person Teams in Urban Search and Rescue [7]. Since that time, it has been turned into a NIST led community supported open source project that provides validated models of robots, sensors, and environments.

Through its usage of UDK, USARSim utilizes the physX phsyics engine [8] 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 most 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.

USARSim was originally based upon simulated environments in the Urban Search And Rescue (USAR) domain. Realistic disaster scenarios as well as robot test methods were created (Figure 1(a)). Since then, USARSim has been used worldwide and more environments have been developed for different purposes. Other environments such as the NIST campus (Figure 1(b)) and factories (Figure 1(c)) have been used to test the performance of algorithms in different efforts [9–11]. The simulation is also widely used for the RoboCup Virtual Robot Rescue Competition [12] and the IEEE Virtual Manufacturing and Automation Challenge [13] and has been applied to the DARPA Urban Challenge (Figure 1(d)).

USARSim was initially developed with a focus on differential drive wheeled robots. However, USARSim's open source framework has encouraged wide community interest and support that now allows USARSim to offer multiple robots, including humanoid robots (Figure 2(a)), aireal platforms (Figure 2(b)), robotic arms (Figure 2(c)), and commercial vehicles (Figure 2(d)). In USARSim, robots are based on physical computer aided design (CAD) models of the real robots and are implemented by specialization of specific existing classes. This sturcture allows for easier development of new platforms that model custom designs.



(a) Aldebaran Robotics Nao. (b) Air Robot AR100B.

(c) Kuka KR60, (d) Kiva Robot.

FIGURE 2. Sample of vehicles in USARSim.

All robots in USARSim have a chassis, and may contain multiple wheels, sensors and actuators. The robots are configurable (specify types of sensors/end effectors for example) through a configuration file that is read at runtime. The properties of the robots can also be configured, such as the battery life and the frequency of data transmission.

The ROS Framework

ROS [1] 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. Examples of ROS applications include Willow Garage's Personal Robots Program [14] and the Stanford University STAIR project [15]. Developers of ROS code are encouraged to contribute their code back to the community and to provide documentation and maintainance of their algorithms.

ROS possesses 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 [16]. 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.

Nomenclature ROS uses the concept of stacks, packages, nodes, messages, topics, and services. These terms are used throughout the rest of the paper and are detailed below [16].

- Node: An executable unit which communicates with other nodes. In this context, the term "node" is interchangeable with "software module". Nodes communicate with each other by passing messages.
- Message: A strictly typed data structure. Standard primitive types (integer, floating point, boolean, . . .) are supported, as are arrays of primitive types and constants. A node sends a message by publishing it to a given topic.
- Topic: A communication channel between two or more nodes. 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 and/or subscribe to multiple topics.
- Service: A remote procedure call defined by a string name and a pair of strictly typed messages: one for the request and one for the response.
- Package: A compilation of nodes that can easily be compiled and ported to other computers. Packages are necessary to build a complete ROS-based robot control system.
- Stack: Packages in ROS are organized into ROS stacks. Whereas the goal of packages is to create minimal collections of code for easy reuse, the goal of stacks is to simplify the process of code sharing. Stacks are the primary mechanism in ROS for distributing software. Each stack has an associated version and can declare dependencies on other stacks. These dependencies also declare a version number, which provides greater stability in development.

THE ROS/USARSIM INTERFACE

USARSim is designed to communicate over an American Standard Code for Information Interchange (ASCII) TCP/IP socket with a host computer. The host computer initiates the socket interface and "spawns" the desired robot into the simulated world that is currently running on the game server. A robot's configuration is controlled by an initialization file that resides on the simulation system's computer. This file controls such aspects as sensor configuration, battery life, and simulated noise models. Please see the USARSim wiki for more information on robot configuration [2]. 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 basic topics to and from 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 platform velocities, and to receive feedback from sensors over other low-level topics. These stacks may also place constraints or naming conventions on the topics. In order to close this low-level loop between ROS and USARSim, a USARSim package was created. This package contains a node called "us-

Parameter	Default	Definition
robotType	P3AT	Type of robot to spawn.
hostname	localhost	Name of host running USARSim.
port	3000	TCP/IP Port on which USARSim listens.
startPosition	Vehicle1	Named location where robot should be spawned. This location is simulated world dependent.
odomSensor	InsTest	Odometry sensor that should be used as the default sensor for feeding the "odom" topic of ROS.

TABLE 1. Parameters for USARSim ROS node.

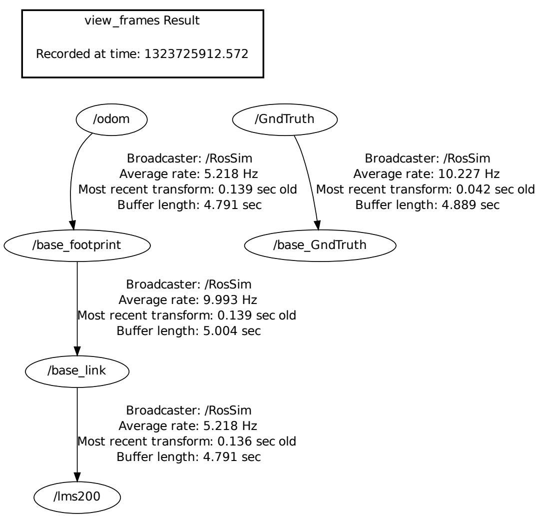


FIGURE 3. Auto-generated tf Transform tree for P3AT robot.

arsim" that publishes a ROS tf transform tree and sensor messages, and also accepts platform and actuator motion commands. When run, it provides a mechanism for spawning a robot in USARSim, and then auto-discovering the robot's sensors, actuators, and drive configuration in order to provide the necessary ROS topics.

The USARSim node relies on several parameters for its configuration. These are detailed in Table 1, and provide information necessary for the creation of a robot in USARSim and a transform tree in ROS. A transfrom tree for the P3AT robot is shown in Figure 3. This transform tree is built automatically from data obtained from the USARSim *geo* and *conf* messages. Since USARSim supports more then one localization sensor on a robot, the *odomSensor* parameter is consulted to determine which sensor should be built into the tree. 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. Additional localization sensors (e.g. the ground truth sensor for the P3AT robot) are provided with their own transform tree.

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 required steering angle and linear velocity. ROS provides a *cmd_vel* topic that includes both linear and angular velocities. The USARSim node automatically converts these velocities into the appropriate commands and values for the USARSim simulator based on the robots steering type, 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 USARSim strives to automatically match simulated sensors to the appropriate ROS topic. Currently, USARSim inertial navigation, ground truth, and LADAR sensors are supported. These sensors automatically join the robot transform tree and publish their sensor messages at the rate that the USARSim node receives the sensor output. Is the intent of the authors to implement the full array of USARSim's sensors as time and resources permit.

The ROS/USARSim interface allows one to utilize known, published algorithms with simulated sensors and environments. However, the computational expense of the sensor processing must still be carried by the target hardware. One benefit of simulation is that one can not only simulate raw sensor sensor output, but also the results from complex sensor processing tasks. One such example is the USARSim object recognition sensor. This sensor is simulated in much the same manner as a laser scanner. However, instead of reporting the range that each beam travels, the sensor accumulates the number of beam hits that occur on each detected object. The number of hits, along with the percentage of the object that is visible may then be used to determine the amount of noise to add to the objects position and recognized type. This information may then be sent over standard ROS topics, without incurring the overhead burden of running the actual object and pose recognition algorithms.

FIGURE 4. Pioneer 3-AT (P3AT) in USARSim.

Mobile Robot Control with the ROS Navigation Stack

Control of mobile robots through the ROS/USARim interface is performed with the ROS navigation stack[2]. The navigation stack is a 2D navigation stack that takes in information from odometry, sensor streams, and a goal pose and outputs safe velocity commands that are sent to a mobile base. The velocity commands are sent it the form of: x velocity, y velocity, theta velocity. Better performance of the navigation stack can be achieved by meeting the following requirements:

- The robot has to use either differential drive or holonomic drive.
- A planar laser has the be mounted on the mobile base. This laser is used for map building and localization.
- The performance of the navigation stack will be best on robots that are nearly square or circular. It does work on robots of arbitrary shapes and sizes, but it may have difficulty with large rectangular robots in narrow spaces like doorways.

Although different models of mobile robot are developed in USARSim, the Pioneer 3-AT (P3AT) (Figure 4) appears to be a suitable candidate to use the navigation stack. The P3AT is a small square-shaped differential wheeled robot with a SICK Laser Measurement Sensor (LMS) 200 mounted on his base. The P3AT is also widely employed for research and prototyping applications involving mapping, navigation, monitoring, reconnaissance, vision, manipulation, cooperation, and other behaviors.

Low-level Navigation The ROS/USARSim interface allows the startup and the control of the default P3AT base controllers by directly sending velocity commands to the base. This task was performed using the following commands:

1. Bring up an environment in USARSim
2. $ `roscore`
3. $ `roslaunch usarsim usarsim.launch`
4. $ `rosrun teleop_twist_keyboard teleop_twist_keyboard.py`

[2]http://www.ros.org/wiki/navigation

In the first step, the user starts the environment in USARSim. If an environment is not up and running, passing messages between ROS and USARSim is not possible. The second step starts `roscore`, a collection of nodes and programs that are a pre-requisites of a ROS-based system. `roscore` must run in order for ROS nodes to communicate. The third step launches the `usarsim.launch` file. This launch file contains information necessary to connect ROS to the computer running the server (USARSim), to set up the appropriate robot (the P3AT in this case) at the correct location in the environment, to launch the proper ROS topics and to start the `RosSim` node. The last step starts the `teleop_twist_keyboard` node which sends velocity commands to the `RosSim` node through the computer keyboard. At this point, the P3AT can be controlled by keyboard teleop in the USARSim environment.

Figure 5 illustrates the communication between the `RosSim` and the `teleop_twist_keyboard` nodes (in ovals). The keyboard inputs are converted in velocity commands and then communicated to the `RosSim` node on the topic `cmd_vel`. `RosSim` publishes two topics for the odometry (`GndTruth` and `InsTest`), one topic to keep track of multiple coordinate frames over time `tf`, and a topic for the laser scanner (`lms200`).

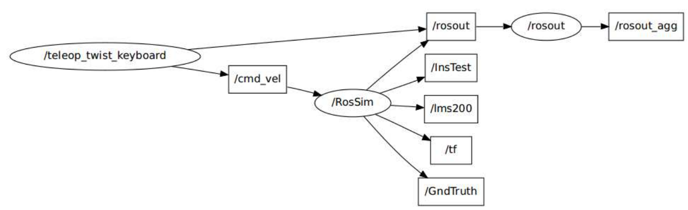


FIGURE 5. Mobile robot control at the lower level.

High-level Navigation This section describes how goals are sent using code to the P3AT to move to a particular location. Navigation at hight level is possible with the action specification for `move_base`. This package provides an implementation of an action (`actionlib`) that, given a goal in the world, will attempt to reach it with a mobile base. The `move_base` node links together a global and local planner to accomplish its global navigation task.

The `move_base` node provides a ROS interface for configuring, running, and interacting with the navigation stack on a robot. The diagram in Figure 6 depicts a high-level view of the `move_base` node and its interaction with other components of the navigation stack. The white components are required components that are already implemented, the green components are optional components that are already implemented, and the blue components must be created for each robot platform.

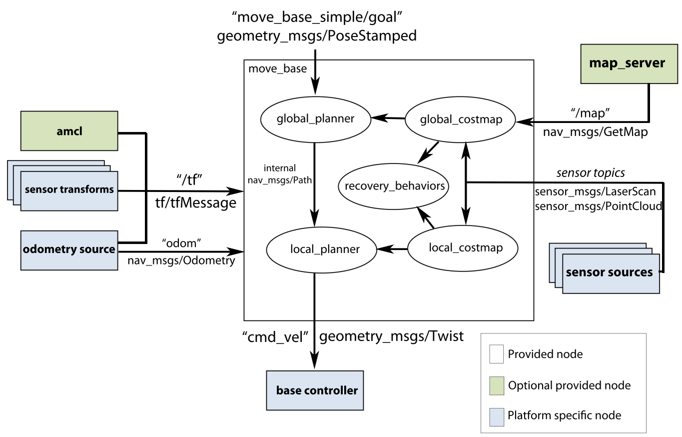


FIGURE 6. Navigation stack setup.

Before running the `move_base` node on the P3AT, localization, mapping and navigation information are filled in the `move_base.launch` file:

- Localization uses map, laser data, and odometry to situate the robot in relation to the environment. The `amcl` and the `map_server` nodes are necessary for robot localization. `amcl` is a probabilistic localization system for a robot moving in 2D and implements the KLD-sampling [17]. The `amcl` node is launched from the examples directory of the amcl package.
- The `map_server` node uses an a priori map generated by the `map_saver` command-line utility. The generated map is stored in pair of files: the YAML file describes the map metadata and a reference to the image file that encodes the occupancy data.
- The navigation stack uses costmaps to store information about obstacles in the world. A global costmap is used for creating long-term plans, and a local costmap is used for local planning and obstacle avoidance. Both costmaps need to follow some configuration options, stored in a third costmap file. Details on the costmaps are stored in YAML files.
- To compute velocity commands to send to the robot given a high-level plan, the navigation stack uses a base local planner. Information on the base local planner is stored in a YAML file which sets configuration options based on the specs of the robot.

Once the `move_base.launch` file is setup with the appropriate configuration options, the `move_base` node can be started by using the following command:

```
$ roslaunch move_base.launch
```

– Need rxgraph file to finish this section –

Robotic Arm Interface

SETUP AND RUN THE INTERFACE

CONCLUSION AND FUTURE WORK

REFERENCES

[1] Garage, W., 2011. "Ros wiki". *http://www.ros.org/wiki*.
[2] USARSim, 2011. "Usarsim web". *http://www.usarsim.sourceforge.net*.
[3] Zaratti, M., Fratarcangeli, M., and Iocchi, L., 2007. *RoboCup 2006: Robot Soccer World Cup X, LNAI*, Vol. 4434. Springer, ch. A 3D Simulator of Multiple Legged Robots based on USARSim, pp. 13–24.
[4] Carpin, S., Wang, J., Lewis, M., Birk, A., and Jacoff, A., 2006. *Robocup 2005: Robot Soccer World Cup IX, LNAI*, Vol. 4020. Springer, ch. High Fidelity Tools for Rescue Robotics: Results and Perspectives, pp. 301–311.
[5] Wang, J., Lewis, M., and Gennari, J., 2003. "A Game Engine Based Simulation of the NIST Urban Search and Rescue Arenas". In Proceedings of the 2003 Winter Simulation Conference, Vol. 1, pp. 1039–1045.
[6] Games, E., 2011. "Unreal development kit". *http://udk.com*.
[7] Lewis, M., Sycara, K., and Nourbakhsh, I., 2003. "Developing a Testbed for Studying Human-Robot Interaction in Urban Search and Rescue". In Proceedings of the 10^{th} International Conference on Human Computer Interaction, pp. 22–27.
[8] Nvidia, 2011. "Physx description". *http://www.geforce.com/Hardware/Technologies/physx*.
[9] Wang, J., Lewis, M., Hughes, S., Koes, M., and Carpin, S., 2005. "Validating USARSim for use in HRI Research". In Proceedings of the Human Factors and Ergonomics Society 49^{th} Annual Meeting, pp. 457–461.
[10] Balaguer, B., Balakirsky, S., Carpin, S., Lewis, M., and Scrapper, C., 2008. "USARSim: a Validated Simulator for Research in Robotics and Automation". In IEEE/RSJ IROS 2008 Workshop on Robot Simulators: Available Software, Scientific Applications and Future Trends.
[11] Kootbally, Z., Schlenoff, C., and Madhavan, R., 2010. "Performance Assessment of PRIDE in Manufacturing Environments". *ITEA Journal*, ***31***(3), pp. 410–416.
[12] RoboCup, 2011. "Robocup rescue homepage". *http://www.robocuprescue.org*.
[13] IEEE, 2011. "Vitrual manufacturing and automation home page". *http://www.vma-competition.com*.
[14] Wyobek, K., Berger, E., der Loos, H. V., and Salisbury, K., 2008. "Towards a Personal Robotics Development Platform: Rationale and Design of an Intrinsically Safe Personal Robot". In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pp. 2165–2170.
[15] Quigley, M., Berger, E., and Ng, A. Y., 2007. *AAAI 2007 Robotics Workshop*, Vol. 85. ch. STAIR: Hardware and

Software Architecture, pp. 6–23.

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

[17] Fox, D., 2003. "Adapting the Sample Size in Particle Filters Through KLD-Sampling". *International Journal of Robotics Research,* ***22***, pp. 985–1003.