

OpenRAVE: A Planning Architecture for Autonomous Robotics

Rosen Diankov

James Kuffner

rdiankov@cs.cmu.edu

kuffner@cs.cmu.edu

CMU-RI-TR-08-34

July 2008

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Copyright © 2008 by Rosen Diankov and James Kuffner. All rights reserved.

Abstract

One of the challenges in developing real-world autonomous robots is the need for integrating and rigorously testing high-level scripting, motion planning, perception, and control algorithms. For this purpose, we introduce an open-source cross-platform software architecture called OpenRAVE, the Open Robotics and Animation Virtual Environment. OpenRAVE is targeted for real-world autonomous robot applications, and includes a seamless integration of 3-D simulation, visualization, planning, scripting and control. A plugin architecture allows users to easily write custom controllers or extend functionality. With OpenRAVE plugins, any planning algorithm, robot controller, or sensing subsystem can be distributed and dynamically loaded at run-time, which frees developers from struggling with monolithic code-bases. Users of OpenRAVE can concentrate on the development of planning and scripting aspects of a problem without having to explicitly manage the details of robot kinematics and dynamics, collision detection, world updates, and robot control. The OpenRAVE architecture provides a flexible interface that can be used in conjunction with other popular robotics packages such as Player and ROS because it is focused on autonomous motion planning and high-level scripting rather than low-level control and message protocols. OpenRAVE also supports a powerful network scripting environment which makes it simple to control and monitor robots and change execution flow during run-time. One of the key advantages of open component architectures is that they enable the robotics research community to easily share and compare algorithms.

Contents

1	Introduction	1
2	Software Architectures for Robotics	1
3	OpenRAVE Architecture	3
3.1	Plugins and Interfaces	5
3.2	Network Protocol and Scripting	7
3.3	Real-time System Interfaces	8
4	Manipulation and Grasping	9
4.1	Barrett WAM	10
4.2	HRP2 Humanoid	10
4.3	Opening Doors with the Manus Arm	10
4.4	Humanoid Walking	10
5	Laser Range Data	11
6	Future Work	13
7	Acknowledgements	14

1 Introduction

As many recent research efforts have shown, creating a robust autonomous robot requires much more than modularity and data flow synchronization [1, 2]. Multiple components of high-level planning, perception, control, error monitoring and recovery, all need to be seamlessly integrated and tested across a wide range of typical input scenarios. The focus of most popular robotics architectures like ROS [3], Player [4, 5], CLARAty [6, 7, 8], and URBI [9] is on modularity, re-usability, communication, concurrency, interoperability, and parallelism. While a few architectures, such as CARMEN [10], Player, and CLARAty, are specifically designed for navigation planning, relatively few architectures include direct support for developing and testing planning algorithms involving complex geometry, such as grasping and motion planning for mobile manipulators or full-body humanoid robots. Moreover, there exists a huge gap between the state-of-the-art in theoretical robotics and practical real-world applications.

We introduce OpenRAVE as an open-source plugin-based planning architecture that addresses some of the limitations associated with currently available robot software. The design of OpenRAVE has been driven primarily by the need for a general-purpose planning and scripting layer to be used in conjunction with low-level robotic packages such as ROS and Player.

Similar to other plugin-based architectures, OpenRAVE can dynamically load modules that conform to its published interfaces, which allows OpenRAVE to grow beyond its original specifications. In some ways, the goals we are aiming to achieve with OpenRAVE are similar to Microsoft Robotics Studio [11] in that both packages provide a 3D physics simulation environment that other modules like robots, sensor systems, and planners can plugin in to provide functionality or to control the flow of execution. However, there are a couple of unique features of OpenRAVE that make it a powerful tool:

- Integral design for real-time control and execution monitoring of robots.
- Core functionality for kinematics operations and physics simulations are provided.
- A network protocol allowing interpreted scripting languages like Octave and Matlab to interface to it (support for additional scripting languages such as Python and Perl are planned for development).
- Built-in core tools and plugin interfaces for manipulation planning and grasping.
- Standard plugins that allow testing of different planning algorithms and sensing systems with minimal code modification.

In this paper we introduce OpenRAVE along with the design process and decisions made in its development. We first give a brief overview of current robotics architectures. We then explain the motivation for OpenRAVE's design in the context of other software packages. Lastly, we illustrate the use of OpenRAVE with several successfully integrated real-world robotic systems, including control of manipulators and humanoid robots.

2 Software Architectures for Robotics

Robotics software architectures have evolved over the years to meet the ever-growing capabilities and demands by developers. Examining the history of this architectural evolution can provide useful insights to help identify the current challenges facing robot development. Many software packages trace their origin to tools whose design structure was motivated by the navigation problem. CARMEN, developed at Carnegie Mellon University (CMU), is an open-source modular architecture using inter-process communication to provide functionality for mobile robots. It supports localization, mapping, SLAM, and provides hardware controllers for various mobile platforms. Player [4, 5], another

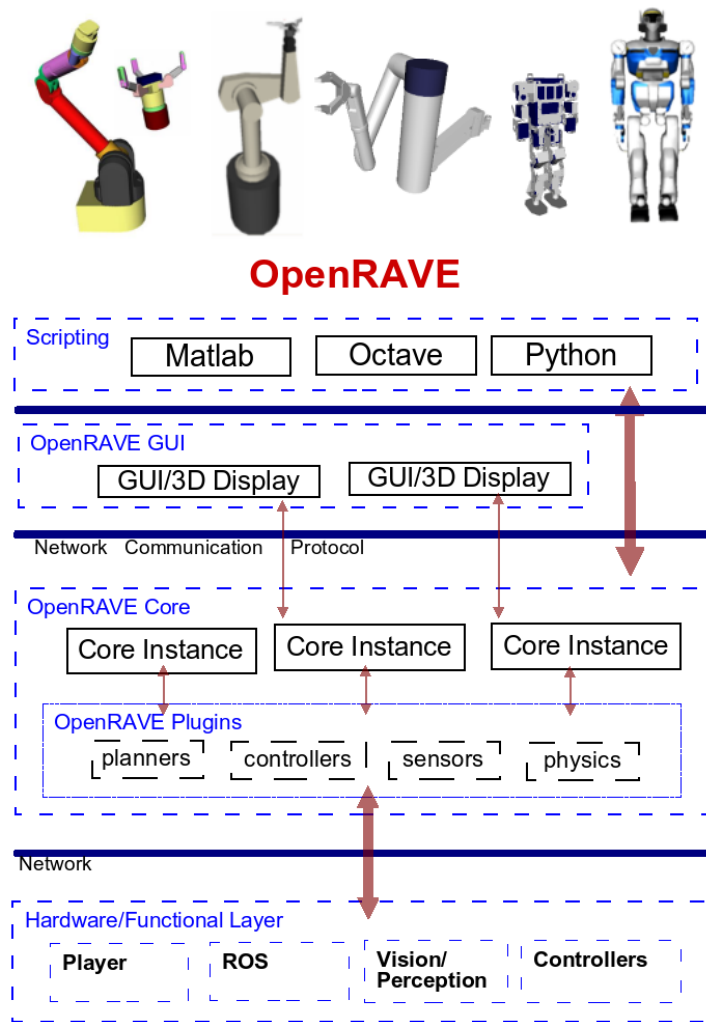


Figure 1: The OpenRAVE architecture is divided into several layers: the scripting layer, the GUI layer, the Core OpenRAVE layer, and the plugins layer that can be used to interact with other robotics architectures.

popular open-source architecture, is similar to CARMEN, except that its design provides additional flexibility. Similar to the way that operating systems drivers work, Player was originally designed to be an API layer between robotics drivers that communicate with hardware and clients that interface with the drivers to move the robot. Soon after the Player project was launched, Stage and Gazebo came out to serve as a simulation component to Player. Because the Player interfaces abstract all hardware, Stage/Gazebo could fool any clients with simulated sensor and controls data, which simplified the migration from simulation to real world implementations. Recently, a new robotics architecture called ROS has been developed to solve some of the problems with the Player client/server model. ROS generalizes every module to a computation graph, and automatically distributes the computation of the modules across the network. One of its main features is the ability to communicate with modules through XML-RPC and specify the data format

before beginning to send data. One of the fundamental goals of ROS is to create robotics software that is platform, language, and implementation independent.

Simulators can play a vital role in the thorough testing of planning and control algorithms for autonomous robots. This was demonstrated by the RAVE simulator [12], the ancestor of OpenRAVE, which was developed for testing humanoid robot motion planning algorithms. Because humanoids require complex controllers for dynamic balance and walking, RAVE enabled rigorous testing of software in simulation prior to execution on the real hardware. OpenRAVE was developed as an open-source replacement to RAVE, but shares much of its original motivation. One notable recent open-source tool designed specifically for humanoid robotics research is OpenHRP [13], which includes tools simulating dynamic walking and control. OpenRAVE is complementary to OpenHRP, as its primary focus is on the development and testing of motion planning algorithms.

As for commercial software, there are two architectures that stand out: CLARAty and Microsoft Robotics Studio. One of the main features of CLARAty, developed by the Jet Propulsion Lab and CMU, is the distinction between the functional and decision layers. The functional layer is analogous to what Player offers: common interfaces to abstract the implementation details of different devices and algorithms. Each module in the functional layer has clearly defined inputs and outputs, and execution is sequential along the functional hierarchy. The decision layer, on the other hand, is given an abstract set of goals and constraints, and its job is to search for the possible actions of the robot to achieve those goals.

The core of Microsoft Robotics Studio is a 3D simulation environment that can load modules written in C#, C++, VB.NET, and Python; it allows users to visually interact with their robots during development. Each module is treated as a service concurrently communicating with other services that can be swapped in and out. Although these packages have been gaining momentum in the past few years, the fact that they are closed-source can limit available execution platforms and the flexibility of new features. For example, adding support for run-time interpretation of scripting languages cannot be done easily without the close cooperation of the package developers.

To summarize: in considering the last decade of progress towards a unified robotics framework, we identify several factors that have contributed to the evolution of architectures and their expectations:

1. The growing role of simulation in the debugging and development cycle.
2. The ability to easily write high-level scripts, as well as utilize more sophisticated development/debugging tools.
3. Separation of modules into *functional* and *decision* layers. The functional layer abstracts the underlying devices, communications, and platform, while the decision layer coordinates high-level behaviors, tasks, and sub-goal selection.
4. Being able to tune parameters and alter execution flow during run-time. This not only provides an efficient means to try out new ideas, but it also makes the development cycle more efficient because fixing a bug does not require a restart of the entire system.

The goal of OpenRAVE is to provide a convenient, open-source framework that provides all of the necessary components for programming robots to perform complex tasks. OpenRAVE unifies not only controls and sensors, but also motion planning algorithms for complex tasks such as those involving dynamic balance, autonomous grasping, and object manipulation among obstacle-cluttered environments.

3 OpenRAVE Architecture

The OpenRAVE architecture modularizes the execution and planning layer of a robotics system so that development of autonomous systems becomes easier and components become more reusable for other projects. One of the fundamental

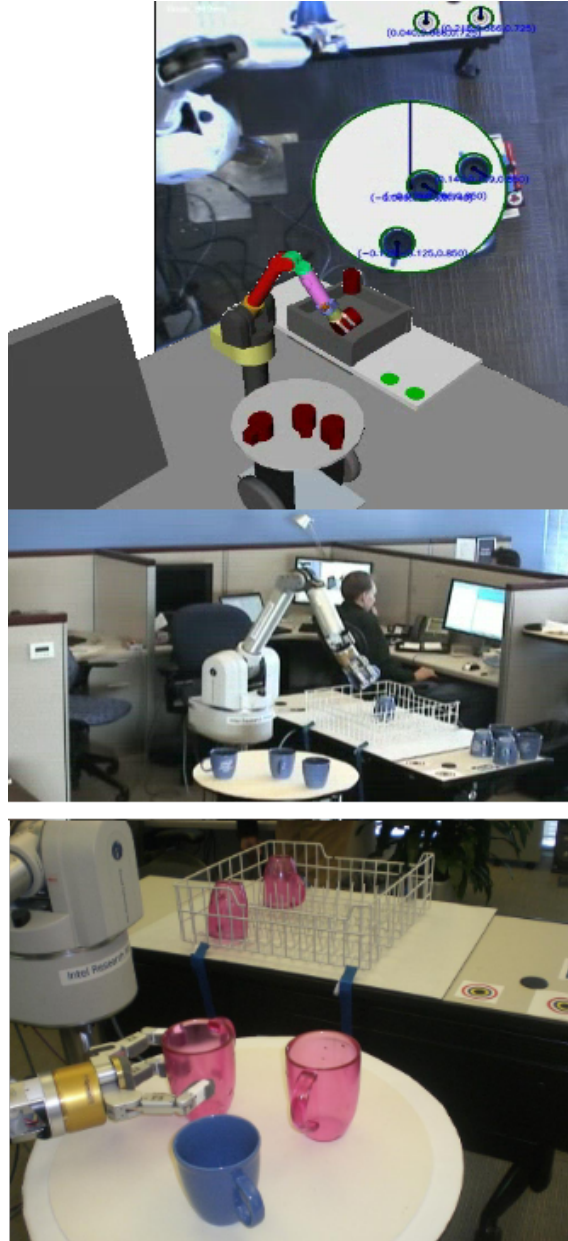


Figure 2: WAM loading cups from a segway to a dishwasher. OpenRAVE is used to sense and plan arm motions in real-time. An overhead camera (*top image*) tracks the segway, cups, and table, and feeds relative pose estimates to the OpenRAVE arm motion planning plugin.

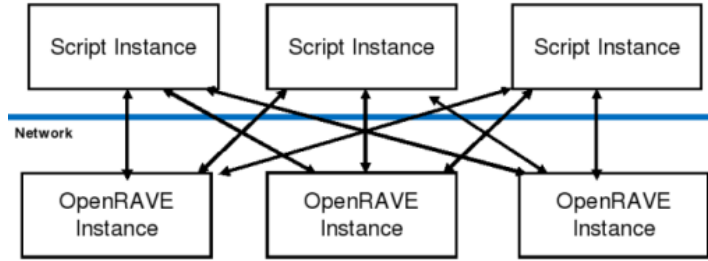


Figure 3: The network protocol allows any script to manage multiple OpenRAVE instances.

decisions was to create an interface layer between the implementation of a specific component and its usage from other components. Many previous architectures have done this for basic level components, such as standardizing formats for laser data, navigation data, and robot control. However, OpenRAVE is unique in that it provides standard interfaces across planners, sensing algorithms, and high level execution modules. Because each of these components are dynamically loaded through plugins, the plugin developer is free to do anything outside the system as long as the interface requirements are met.

The OpenRAVE architecture is divided into four distinct layers (Figure 1). At startup, the Core starts a server, which manages and updates its own environment. It can continuously publish all objects currently loaded and accepts commands that can load plugins, invoke planners, send control commands to robots, or add new objects. Each Core instance is not bound to any particular graphical user interface (GUI) or scripting session; in fact, the scripts and GUIs are separate processes that connect to any Core instance and initialize communications with it. Currently, all GUI/Script/Core connections are implemented through TCP/IP, but support for communication via shared memory is planned. Scripts are meant to be used to control the overall execution, while user interfaces are used for visualization and debugging.

Because terminating and restarting a script does not require a restart of a Core instance, scripting also provides an ideal way to debug a system and store parameters or other heuristics that may require tuning. The GUI instance spends most of its time querying OpenRAVE for the most recent pose estimates of scene objects so it can display a consistent world view of internal state. Because the GUI uses the same client/server model as the scripts, the user can interact with a scene object through the GUI and seamlessly change the internal OpenRAVE state for that object. In addition, the client/server model allows any scripting or GUI instance to simultaneously communicate with multiple OpenRAVE Core instances (Figure 3); and the Core instances can communicate with each other to periodically synchronize their internal view of the world.

Once a Core instance is started and all plugins are loaded, OpenRAVE will then expose the functionality of its loaded plugins. Therefore, the best way to communicate with OpenRAVE plugins from external package is to implement the network protocol interface in the native package (like scripts do).

3.1 Plugins and Interfaces

OpenRAVE identifies specific interface categories that can be implemented by the plugin.. The interface types are:

- **Planners** - A plan is a trajectory or a policy that the robot has to follow in order to get from its initial state to a goal state while maintaining certain constraints, such as maintaining dynamic balance or avoiding collisions with obstacles. Planners produce plans from initial conditions.
- **Controllers** - Every robot is attached to a controller used to move it in the world (simulated or real). The controller provides functionality to get or set a trajectory and query the robot's current state.

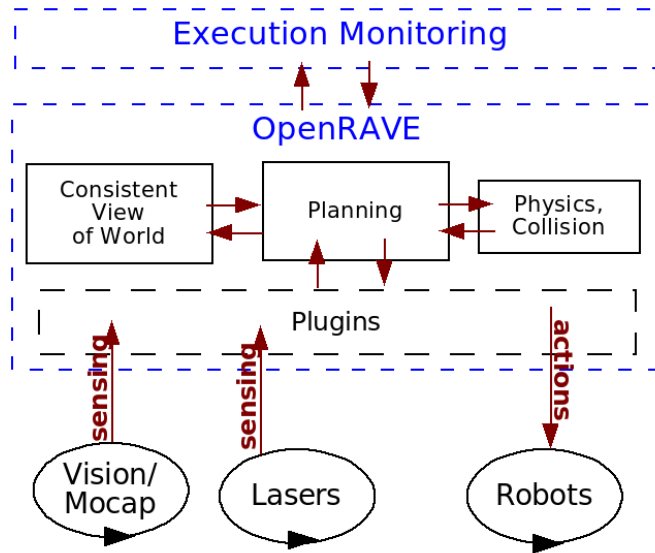


Figure 4: The path flow for real-time execution.

- **Sensors** - A sensor, like a range finder or a camera, gathers information about the world and returns it in a standard format. A sensor can be attached to any part of a robot.
- **SensorSystems** - A system that can update object pose estimates arbitrarily due to some external input devices, such as a motion capture system, vision cameras, or laser range data.
- **Problem Instances** - Each problem instance is analogous to a small program embedded in OpenRAVE. Once created, a problem instance is registered with the main simulation loop and the OpenRAVE network server through the *SendMessage* function. Problem instances can offer special functions for manipulation or navigation and can easily expand the network capabilities of OpenRAVE.
- **Robots** - OpenRAVE supports a variety of different kinematic structures for robots with unique capabilities. For example, interfaces for a humanoid robot can differ significantly from interfaces for wheeled mobile robots. Providing implementations for various classes of robots enables clients to better exploit their structure.
- **Inverse Kinematics Solvers** - IK solvers can be specified and return closed-form or numerical solutions that can be used as input to manipulation planners. Each IK solver can be attached to a subset of the links of a robot.
- **Physics Engines** - OpenRAVE provides the capability to use any custom simulation system library through a plugin without requiring any other plugin to know details of the library or how it is linked.

The current OpenRAVE model loads all plugins in the same Core process. In terms of compatibility, this is less flexible than having each plugin as a stand-alone process communicating with the Core across yet another layer. However, the rationale behind this design decision is that some messages and function calls, such as collision querying, may need to be called at a high frequency (e.g. thousands of times per second). For example, a motion planner may invoke a call to the **CheckCollision** function for every candidate robot configuration during the search. Although it is possible to use the network protocol to perform these queries, jumping to a memory address is much more efficient

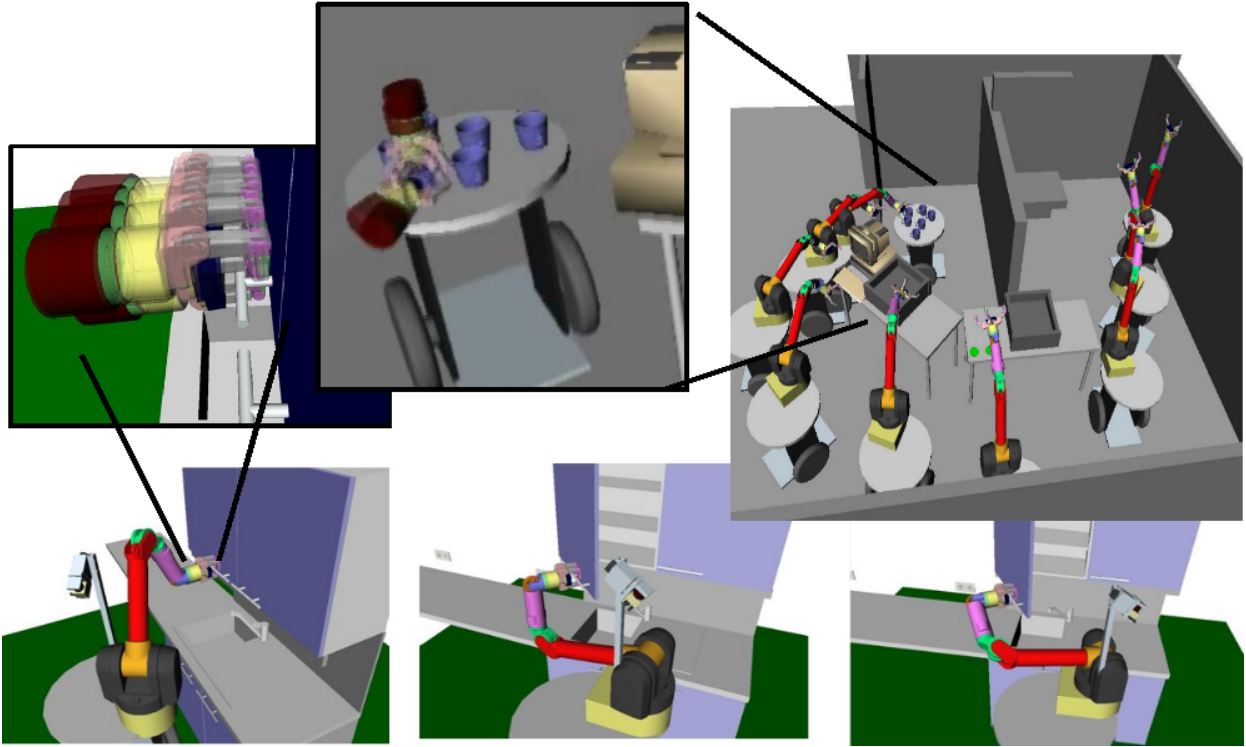


Figure 5: Manipulation and grasping experiments with the WAM. For opening a cabinet, the WAM uses a caging grasp set to increase its manipulability range (*top left*). Planning pick and place operations for the WAM mounted on a mobile base (*top right*). The planner simultaneously evaluates a large sampling of possible grasps for the cup object (*top center*).

than utilizing TCP/IP through shared memory. Similarly, complex kinematics or physics simulation queries executed in a tight loop require that the implementation be abstracted from the caller, and the query itself be as fast as possible. Given these considerations, the design decision to load all plugins in the Core process is not as strict as it initially seems. Planners and other functions are still able to run on multiple threads and plugins can connect to other systems through the network or shared memory as needed. Furthermore, multiple Core instances can exist, each loading its own set of plugins and running its own set of planners.

3.2 Network Protocol and Scripting

To briefly summarize the OpenRAVE network server, the available commands provide the following services:

- Communication with robot controllers.
- Reading the state estimates of scene objects, robot joint values, link transformations, and link geometry.
- Setting object poses and robot joint angle values.

- Performing object-object or ray-object collision queries.
- Creating or destroying any robot, object, environment, or problem instance.
- Sending commands to problem instances and obtaining their results.
- Plotting point clouds, lines and other primitives within the OpenRAVE 3D environment visualization GUI.
- Loading and reloading plugins.
- Setting debug modes, and tuning planning, simulation, and GUI parameters.

In the current implementation, the network commands are sent through TCP/IP and are text-based. The text-based commands allow easy interpretation of the data and make supporting a scripting language straightforward. The protocol is not limited to network and TCP/IP connections. In the future, we plan to implement an arbitration layer through XML-RPC similar to ROS[3] that can decide the best format and method to transmit data across the Scripting, GUI, and Core layers. For example, scripts running locally to a Core instance should automatically utilize shared memory to communicate between the two processes.

OpenRAVE currently supports Octave and Matlab scripting environments that communicate via network sockets. Interacting with the OpenRAVE functions is seamless. Users need only to set the IP address of the host computer running OpenRAVE, and all the other details will be handled.

3.3 Real-time System Interfaces

Aside from providing a convenient testbed for simulation and algorithm development, OpenRAVE was designed with interfaces for real-time control of a robotic system. Any algorithms and scripts designed to work within a simulated OpenRAVE world can in theory operate in the real world simply by replacing simulated robot controllers with real ones and connecting a sensor system instance for real-time feedback. In practice, some effort is generally required to tune algorithms and meet performance requirements when executing on a real robot. Mindful of this, OpenRAVE's design attempts to minimize manual effort required in porting a working simulation to a real-world system.

Many real-time robotic systems include a "Sense-Plan-Act" loop encapsulated within an execution monitor [1, 14]. Figure 4 shows one example of how this loop could be organized within the framework of OpenRAVE.

The "robotic busboy" [15] is an example of a real-time system that uses such an execution loop within OpenRAVE (Figure 2). Here, the robot's task is to autonomously pick up cups from a tray mounted on a segway and place them in a dish-rack.

Because the segway is free to move and the tray can be cluttered with mugs in arbitrary positions, the robot arm motion planner must consider many potential stable grasps of each mug in order to decide the best strategy to clear the tray successfully. The vision server and robot controllers are implemented as Player drivers, and the appropriate Player clients are OpenRAVE plugins. Finding the grasps and performing the planning is implemented by a grasp planner and a Bidirectional RRT according to [16]. A manipulation problem instance is created to accept simple **move-to** commands from the scripts and to make robot plans. The execution monitor and overall task management is managed by a Matlab script. The script first enables the vision server to update the objects in the environments. Then grasp selection heuristics are used to determine an ordering for picking up the mugs on the tray. The script then performs a series of checks and validations in order to plan a simulated sequence of paths to move the robot arm from its current state and complete the subtasks of placing each mug in the dishrack. If simulation succeeds, the planner begins executing the first part of the plan. The script carefully monitors execution for failure conditions, such as unsuccessful grasps or environment changes, and replans as necessary.

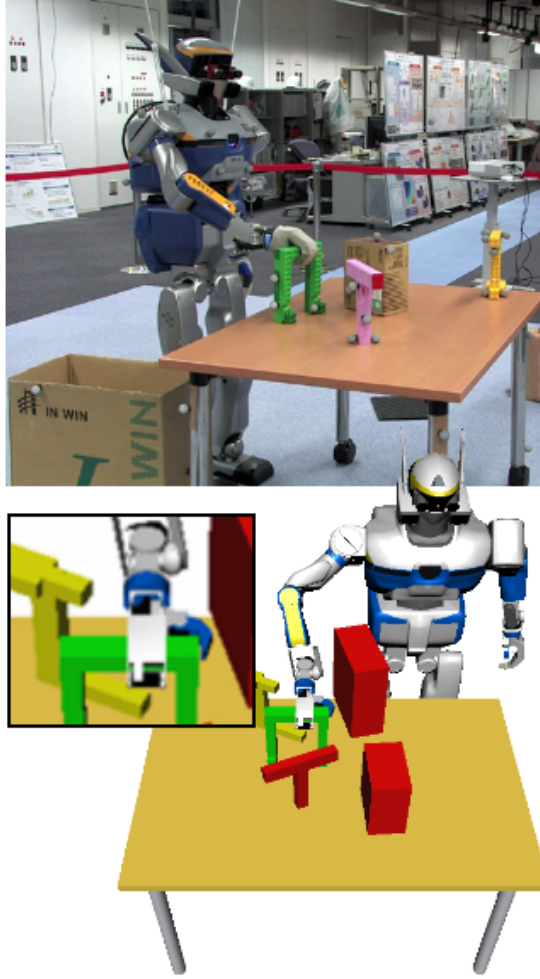


Figure 6: The HRP2 humanoid planning autonomous grasping and manipulation motions to clear assorted objects from a table and place them into a box.

4 Manipulation and Grasping

Although the OpenRAVE architecture can support a variety of robotic tasks, its original design objective was to handle autonomous manipulation tasks. The initial development efforts focused on grasping objects, such as calculating contact points and normals between objects and end-effectors, and computing force closure and grasp stability metrics. Other software tools, notably the GrasPlt! simulator [17], perform similar computations for grasp analysis. OpenRAVE soon expanded to include path planning algorithms and scripting functionality to meet more general planning and manipulation demands [16, 18]. The following sections give a brief overview of four different experimental real-world systems developed using OpenRAVE.

4.1 Barrett WAM

Most grasping research considers free-floating end-effectors able to approach a target object from any direction. In the real world, end-effectors are typically attached to manipulator arms whose motion is subject to both kinematic and dynamic constraints. Therefore, to solve the simple problem of autonomously picking up an object from a cluttered environment, it is necessary to at least consider not only the end-effector and target object shapes, but also the surrounding environment obstacles and the motion capabilities of the arm [16]. The problem becomes even more difficult when the arm is attached to a mobile base because computing arm capability, such as target reachability may involve more than simple inverse kinematics [18]. Figure 5 shows several autonomous grasping and manipulation planning capabilities implemented on a Barrett WAM arm and Segway mobile base. OpenRAVE was used to setup the task execution framework, develop and test the planning algorithms in simulation, and then manage the execution of the planners and scripts on the actual robot hardware. We believe this process helped enable the porting from simulation to the real robot to take only a few weeks rather than several months of rewriting and tuning software.

4.2 HRP2 Humanoid

The same framework as used for the Barrett WAM has been applied to the HRP2 humanoid robot [16] (Figure 6). Because only the high-level scripts contain robot-specific parameters, the lower-level plugins developed for grasp generation and arm motion planning are fairly clean and easily generalizable to different hardware platforms. The only differences between the Barrett setup and the HRP2 setup involved the robot controller and the sensor system plugin. The HRP2 setup used a commercial motion capture system for tracking object motion, while the Barrett setup used a single ceiling-mounted camera. In both cases, all implementation-specific details are hidden within the **sensor system** and **controller** interfaces, while the planning and grasp generation algorithms did not have to be modified.

OpenRAVE is also useful for performing kinematic analyses to assist in planning for mobile bases (Figure 7). The reachability region can be calculated for a manipulator, which encodes the volume of reachable space for an end-effector relative to the robot base. For planning mobile manipulation, the *inverse reachability map* can be used to compute possible base placements of the robot given a target object for grasping. Once the inverse map is calculated [18], the robot can efficiently determine sets of feasible hand, arm, and body configurations for the robot to grasp a desired object.

4.3 Opening Doors with the Manus Arm

A very useful class of constrained manipulation tasks for robots includes opening drawers, cabinets, and doors. Two major sources of difficulty are dealing with *kinematic redundancy* and *task compliance*. These two problems are intertwined because achieving task compliance may require kinematic redundancy in order to ensure sufficient degrees of freedom (DOF) to successfully perform a constrained task. Using OpenRAVE, it is possible to compute special *caging* grasp sets for doors and other constrained objects (Figure 5). Caging grasps are more tolerant of execution errors than fixed, rigid grasps, as compliance is inherent. Planning door-opening motions using a set of constrained caging grasps greatly increases the allowed motions of the robot arm and base. Figure 8 shows a six DOF Manus robot arm mounted on a wheelchair base computing a strategy to open a door more than 60 degrees, a task which cannot be solved if only fixed rigid grasps are considered.

4.4 Humanoid Walking

Besides manipulation, OpenRAVE has also been used to generate walking gaits for the small-size humanoid robot “Choromet” (Figure 9). OpenRAVE does not yet come with sophisticated dynamic walking controller plugins, and the

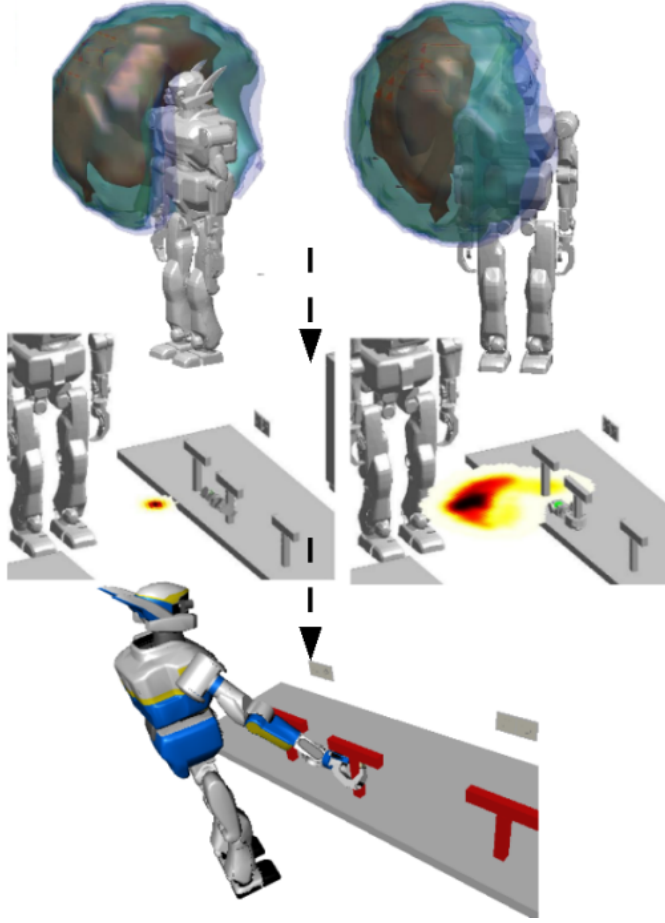


Figure 7: Mobile manipulation with the HRP2 humanoid robot. The IK reachability region for HRP2’s right hand (*top*). The inverted reachability map illustrating candidate body positions for HRP2 (*center*). Final selected grasp including arm configuration and body pose (*bottom*).

dynamic simulation framework has not been applied to advanced research topics such as walking on rough terrain or pushing heavy objects [19]. However, the framework was designed to support humanoid balance and walking research, and we expect the capability and number of available plugins to increase as more researchers experiment and utilize open-source tools like OpenRAVE and OpenHRP [13].

5 Laser Range Data

We briefly summarize another application involving using OpenRAVE to help simulate, test, and debug a sensor subsystem. In this example, experiments were conducted to evaluate placement, accuracy, and use of laser rangefinding equipment in planning manipulations for the Barrett WAM arm system introduced in the previous section. Figure 10

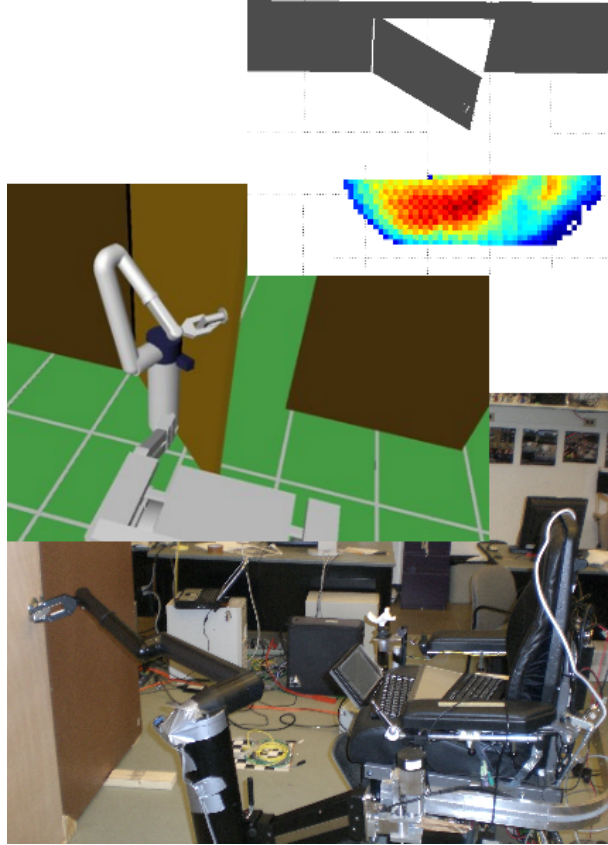


Figure 8: A highly-constrained 6-DOF wheelchair-mounted Manus arm opening a door by caging the door handle. A performance map (*top*) illustrates an overhead view of possible placements for the wheelchair base, with red indicating base center positions that allow the robot to open the door further.

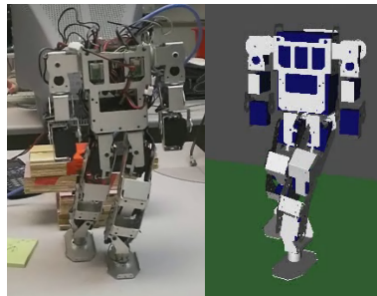


Figure 9: The small-size Choromet humanoid robot executing a statically-stable walking gait.

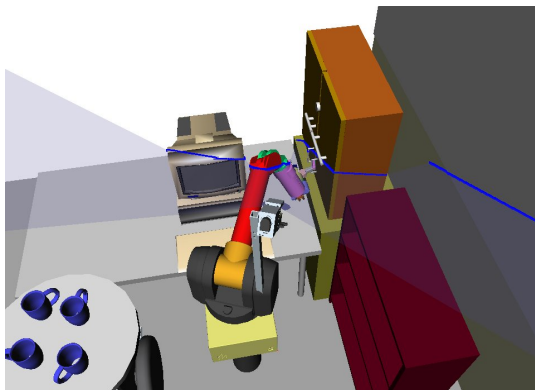


Figure 10: Simulation of a laser range finder mounted on a WAM arm. Blue dots indicate laser hit points and blue transparent plane is the laser plane.

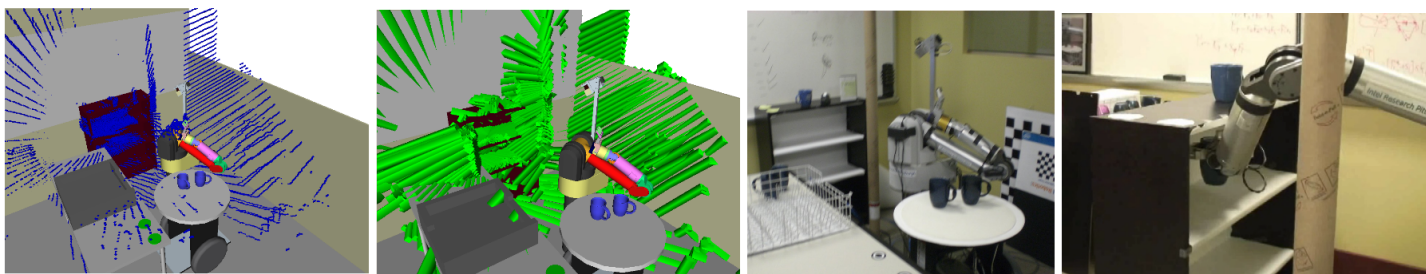


Figure 11: Laser data scans are aggregated into bounding collision obstacle cylinders. The cylinders are pruned based on proximity to the WAM arm. Finally, the planner computes a collision-free path for the arm to place cups on the shelf while avoiding the sensed obstacles.

illustrates simulated range data and processing within OpenRAVE. After evaluation, OpenRAVE was used to process actual laser data in real-time and aggregate obstacle surface collision models, which the robot arm should avoid (Figure 11). In the future, we plan to simulate additional physical device characteristics, such as adding models of sensor-delay, noise, and uncertainty.

6 Future Work

In this paper we have introduced an open-source architecture that concentrates on planning algorithms as well as real-time control of robots. OpenRAVE is still a work in progress, and future updates will add new features to make robot development and integration with other robotics packages easier. There are several major areas of improvement we are considering at the moment. The first is standardizing the network protocol and possibly sharing messages with ROS for a more seamless integration. This will still keep OpenRAVE as a stand-alone package. The second area is to concentrate on parallel execution of planners and real-time control. The ability to plan ahead for future intermediate goals can increase performance through pipelining planning and execution. A third focus area is to experiment further with execution monitors and execution models, and ways to integrate these within OpenRAVE. In particular, execution

monitoring can be vital for failure recovery, robustness in uncertain environments, and reliably performing more complex tasks.

In conclusion, we reiterate our belief in the importance of open-source robotics architectures for standardizing formats and development paradigms. With OpenRAVE, we are also aiming to build a repository of planning algorithms and develop standardized test cases for them. One of the challenges to motion planning research is that it is very difficult to compare two different algorithms, because they may have been developed on different platforms and tested on a different set of problems. By having a repository of real-world problems, planning researchers can gather statistics more easily and at the same time build up a collection of standard planners within the framework of OpenRAVE.

OpenRAVE and plugins are available through the OpenRAVE Wiki at (<http://openrave.programmingvision.com>) and SourceForge (<http://sourceforge.net/projects/openrave>).

7 Acknowledgements

We would like to thank Dmitry Berenson for his invaluable suggestions and contributions to parts of OpenRAVE. We thank the Digital Human Research Center (AIST) for collaborating on experiments with the HRP2 humanoid robot. We also thank Intel Research Pittsburgh their continued support and collaboration with the WAM arm and Segway. Partial support for this research was provided by the Quality of Life Technology Center under NSF grant EEC-0540865.

References

- [1] C. U. W. Whittaker, "Self-driving cars and the urban challenge," *IEEE Intelligent Systems*, vol. 23, no. 2, pp. 66–68, 2008.
- [2] R. Simmons, S. Singh, F. W. Heger, L. M. Hiatt, S. Koterba, N. Melchior, and B. Sellner, "Human-robot teams for large-scale assembly," in *Proceedings of NASA Science Technology Conference*, 2007.
- [3] "<http://pr.willowgarage.com/wiki/ros>."
- [4] B. Gerkey, R. T. Vaughan, K. Stoy, A. Howard, G. S. Sukhatme, and M. J. Mataric, "Most Valuable Player: A Robot Device Server for Distributed Control," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2001.
- [5] T. Collett, B. MacDonald, and B. Gerkey, "Player 2.0: Toward a practical robot programming framework," in *Australasian Conference on Robotics and Automation*, 2005.
- [6] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das, "The claraty architecture for robotic autonomy," in *Aerospace Conference*, 2001.
- [7] I. Nesnas, R. Simmons, D. Gaines, C. Kunz, A. Diaz-Calderon, T. Estlin, R. Madison, J. Guineau, M. McHenry, I. Shu, and D. Apfelbaum, "Claraty: Challenges and steps toward reusable robotic software," *International Journal of Advanced Robotic Systems*, vol. 3, no. 1, 2006.
- [8] "<http://claraty.jpl.nasa.gov/man/overview/index.php>."
- [9] "<http://www.urbiforge.com>."
- [10] "<http://carmen.sourceforge.net/home.html>."
- [11] "<http://www.microsoft.com/robotics>."
- [12] J. Kufner, S. Kagami, M. Inaba, and H. Inoue, "Graphical Simulation and High-Level Control of Humanoid Robots," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2000.
- [13] F. Kanehiro, H. Hirukawa, and S. Kajita, "OpenHRP: Open Architecture Humanoid Robotics Platform," *International Journal of Robotics Research*, vol. 23, no. 2, p. 155, 2004.
- [14] P. Michel, J. Chestnutt, J. Kuffner, and T. Kanade, "Vision-guided humanoid footstep planning for dynamic environments," in *Proceedings of IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2005.
- [15] S. Srinivasa, D. Ferguson, M. V. Weghe, R. Diankov, D. Berenson, C. Helfrich, and H. Strasdat, "The robotic busboy: Steps towards developing a mobile robotic home assistant," in *Proceedings of the International Conference on Intelligent Autonomous Systems (IAS)*, 2008.
- [16] D. Berenson, R. Diankov, K. Nishiwaki, S. Kagami, and J. Kuffner, "Grasp planning in complex scenes," in *Proceedings of IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2007.
- [17] A. Miller, "Graspit!: A versatile simulator for robotic grasping," Ph.D. dissertation, Columbia University, 2001.

- [18] R. Diankov, N. Ratliff, D. Ferguson, S. Srinivasa, and J. Kuffner, “Bispace planning: Concurrent multi-space exploration,” in *Proceedings of Robotics: Science and Systems (RSS)*, 2008.
- [19] M. Stilman, K. Nishiwaki, and S. Kagami, “Learning object models for whole body manipulation,” in *Proceedings of IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2007.