Computer Science Education
Vol. 00, No. 00, May 2013, 1–12

To appear in Computer Science Education,
Special Issue on Robotics in CS Education.

ORIGINAL ARTICLE

Software for Project-Based Learning of Robot Motion Planning

Mark Moll[a]*, Janice Bordeaux[b], and Lydia E. Kavraki[a]

[a] *Rice University, Department of Computer Science, Houston, TX, USA*;
[b] *Rice University, School of Engineering, Houston, TX, USA*

(*Received 00 Month 200x; final version received 00 Month 200x*)

Motion planning is a core problem in robotics concerned with finding feasible paths for a given robot. Motion planning algorithms perform a search in the high-dimensional continuous space of robot configurations and exemplify many of the core algorithmic concepts of search algorithms and associated data structures. Motion planning algorithms can be explained in a simplified two-dimensional setting, but this masks many of the subtleties and complexities of the underlying problem. We have developed software for Project-Based Learning of motion planning that enables deep learning. The projects that we have developed allow advanced undergraduate students and graduate students to reflect on the performance of existing textbook algorithms and their own variations on such algorithms. Formative assessment has been conducted at three institutions. The core of the software used for this teaching module is also used within the Robot Operating System (ROS), a widely adopted platform by the robotics research community. This allows for transfer of knowledge and skills to robotics research projects involving a large variety robot hardware platforms.

Keywords: robotics education; motion planning; project-based learning; educational software

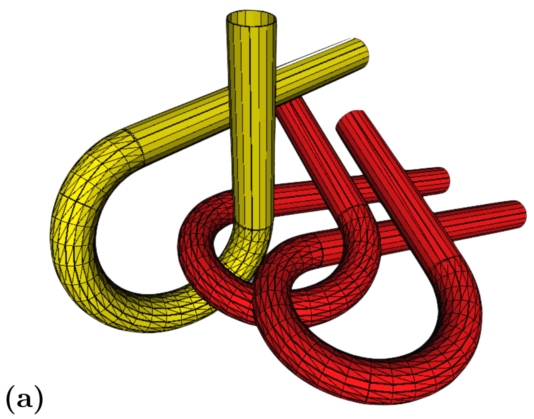
1 Introduction

Motion planning is a key part of robotics: it is concerned with the problem of finding a feasible path for a given robot between a start and goal location. The feasibility of a path is often defined by requiring that the robot not collide with any obstacles, but it can be generalized to include any constraints such as force limits, velocity limits, etc. Two examples are shown in Fig. 1. In introductory robotics classes, motion planning is often simplified to a discrete grid search or reactive navigation strategies that sacrifice completeness and optimality. The state of the art in motion planning has strong algorithmic underpinnings and fits well in a CS curriculum. It requires an understanding of search algorithms for high-dimensional continuous spaces, computational geometry, and some basic topology. Robotics makes abstract concepts in these areas very concrete and makes them easier to visualize. The focus of our project is to provide an intermediate-level curriculum that supports the acquisition of concepts in algorithmic robotics through motion planning projects.

Many practical motion planning algorithms have been proposed that rely on random sampling of robot configurations, but still provide strong theoretical guarantees. Although most of the concepts used in these algorithms can be taught at an abstract level, evidence emerging in engineering educational research indicates that diverse students achieve a deeper understanding when theories and concepts are *inferred from particular problems* (Barron et al., 1998; Prince & Felder, 2006; Graham & Crawley, 2010). Specifically, research on project-based learning and problem-based learning suggests that assigning a series of projects designed to help students infer motion-planning concepts may promote skill development, conceptual knowledge, knowledge

*Corresponding author. Email: mmoll@rice.edu

ISSN: 0899-3408 print/ISSN 1744-5175 online
© 2013 Taylor & Francis
DOI: 10.1080/0899340YYxxxxxxxx
http://www.informaworld.com



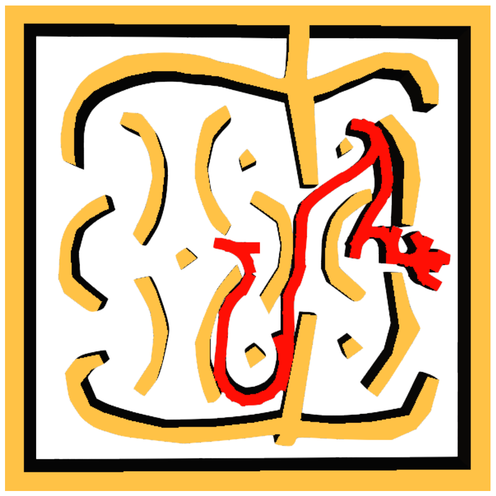


Figure 1. Two sample motion planning problems. **(a)** The alpha puzzle: the red alpha shape needs to be moved from its initial interlocked pose to a pose where it is separated from the yellow alpha shape. Here the yellow alpha shape can be regarded as an obstacle. **(b)** A solution path for a car-like robot in a maze. The "car" (modeled by a red box) cannot move sideways and has a bounded turning radius.

transfer, and metacognitive reasoning (Kay et al., 2000; Gijbels et al., 2005; Chung & Chow, 2004; Galand & Frenay, 2005).

We have adopted a project-based learning (PBL)[1] approach to our curriculum using a widely-accepted definition of PBL provided by Prince & Felder (2006):

> Project-based learning begins with an assignment to carry out one or more tasks that lead to the production of a final product—a design, a model, a device or a compute simulation. The culmination of the project is normally a written and/or oral report summarizing the procedure used to produce the product and then presenting the outcome.

Problem-based and project-based learning are very similar, but in project-based learning the instructional designer exerts great control over the choice of projects. Using a project-based approach helps maintain a focus on the course objectives, while allowing students enough autonomy to formulate their own strategies and solutions (Prince & Felder, 2006).

Motion planning is an ideal candidate for PBL. Many variations of the basic motion problem exist that can provide challenges to students at any level. Also, the complexity of search in high-dimensional continuous spaces is hard to convey verbally in lectures. This is difficult because it involves a search over all the degrees of freedom of a robot (that is, the set of all parameters needed to specify a configuration of a robot) and the students often lack good visual intuition for search in high-dimensional spaces. However, a solution to a motion planning problem can be visualized by having a robot move through a 2D or 3D workspace and having students inspect the path for correctness. This visualization process is highly intuitive (mostly without conscious reasoning).

Our general interest is in robotics education embedded within a CS curriculum. In introductory robotics classes, motion planning is often simplified to a discrete grid search or reactive navigation strategies that sacrifice completeness. As David Touretzky observed (Touretzky, 2010), robotics in the CS context is not about building robots and should not be limited to programming simple reactive behaviors. While great strides have been made in creating affordable hardware platforms to teach students about robotics (Balch et al., 2008), cost is still an issue for many institutions and the algorithmic complexity of the problems that can be investigated is often limited. In recent years, though, many robotics algorithms have been implemented and integrated in large systems that can run both on real hardware and in simulation. The Robotics Operating System (ROS) (Quigley et al., 2009) is a prime example. Robots ranging from simple mobile platforms

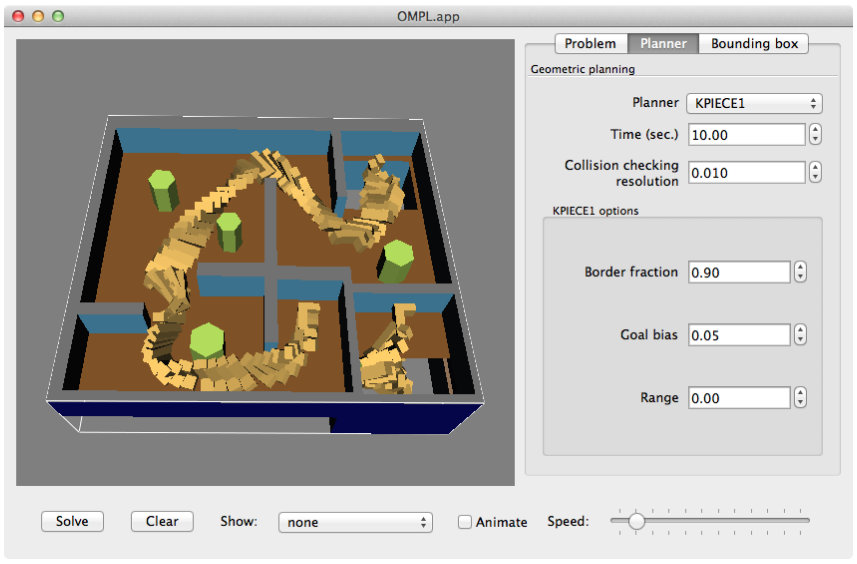
[1]Both problem-based and project-based learning are abbreviated as PBL in the literature. In this article PBL will refer to project-based learning.

to full humanoids can now be studied within the same software framework and in physically realistic simulations. Our Open Motion Planning Library (OMPL) software can be used with or without ROS, and the teaching module we have designed focuses on algorithmic aspects of robot motion planning. As a result, our educational software provides the CS curriculum with an attractive alternative to software created solely for educational purposes. The OMPL software can also be used for educational purposes, but is is advanced enough to be used, maintained, and sustained in the long run by robotics researchers through open source efforts. This dynamic relationship will keep our project-based learning curriculum current, adaptable (depending on hardware availability at an institution) and motivating to students.

1.1 *Motion Planning Concepts*

Motion planning is a mature field covered extensively in several robotics textbooks (Choset et al., 2005; Latombe, 1991; LaValle, 2006). Two of these textbooks concentrated entirely on motion planning (LaValle, 2006; Latombe, 1991), while in the third it occupies almost half the book (Choset et al., 2005). One of the authors (Kavraki) has taught an algorithmic robotics class for many years and is a co-author of one of the main robotics textbooks (Choset et al., 2005). The basic problem of finding collision-free paths for polyhedral robots operating in a polyhedral workspace is already PSPACE-complete (Canny, 1988). Complete planning algorithms are difficult to implement and computationally intractable. Research efforts have therefore shifted to algorithms that provide weaker completeness guarantees. Sampling-based algorithms in particular have emerged as a practical approach to many hard motion planning problems (Choset et al., 2005; Tsianos et al., 2008). The fundamental idea of sampling-based motion planning is to approximate the connectivity of the (continuous) search space with a graph structure. The search space is sampled in a variety of ways, and selected samples end up as the vertices of the approximating graph. Edges in the approximating graph denote valid path segments. Many different sampling-based planners have been proposed, but they all operate using similar concepts (for a survey look at recent papers such as (Şucan & Kavraki, 2012)). The isolation of the relevant concepts is important for educational purposes as it teaches students good algorithmic design and good software engineering practices. The most important concepts are listed below:

- **State space** Points in the state space fully describe the state of the system being planned for. For a free-flying rigid body, for instance, the state space consists of the space of all translations and rotations. Here, topology is very important. For example, in 2D orientations, the points 0 and 2π are identified so that the distance between them is 0.
- **Sampler** A sampler is responsible for producing different states from the state space. While sampling uniformly works well in many cases, one can often do better by recognizing that fewer samples are needed in wide open free spaces than in narrow passages. Here, the topology of the underlying state space is also important. For instance, it is not completely trivial to sample uniformly over the space of 3D rotations.
- **State validity checker** A state validity checker is a function that computes whether a sampled state is valid. For example, it can determine whether the state is collision-free and whether velocities and accelerations are within bounds.
- **Local planner** The local planner usually performs some kind of interpolation in the state space between two different states and checks whether intermediate states are valid.
- **State propagator** For systems with differential constraints such as the car-like robot in Fig. 1(b) there is a function called a state propagator that computes the trajectory when a control is applied for some period of time starting from some initial state. Using numerical integration, a state propagator function can automatically be constructed from a set of Ordinary Differential Equations (ODE) of the form $\dot{q} = f(q, u)$, where q is a vector describing a robot's configuration and u is a control input vector. A planner can use a state propagator to find a trajectory between a start and goal state that consists of a sequence of controls and corresponding durations.



Planner KPIECE1
Time (sec.) 10.00
Collision checking resolution 0.010
Border fraction 0.90
Goal bias 0.05
Range 0.00
Show: none

Figure 2. The OMPL.app GUI showing a path for an L-shaped robot in an office-like environment with two floors and a staircase joining them.

The motion planning curriculum we have developed enhances deep understanding of these concepts as well as several state-of-the art motion planning algorithms that build on these concepts. The software component of the curriculum encourages good software engineering practices.

1.2 *Software Overview*

The concepts described above have been implemented in a software package called the Open Motion Planning Library (OMPL, Şucan et al. 2012), which we released in 2010 under the BSD license (one of the most permissive Open Source licenses). The software is available from http://ompl.kavrakilab.org. The core library is written in C++ with a clear mapping between the object-oriented class hierarchy and the concepts used in the motion planning literature. Almost all the functionality is also accessible through Python bindings. Many institutions use Python for introductory programming classes, so Python bindings are important to reach a large audience. At the same time, for performance reasons we could not implement all low-level functionality in Python. The software can be installed through standard package managers for Linux and OS X, but can also be built from source on MS Windows, OS X, and Linux. In 2012, OMPL won the Grand Prize in the Open Source Software World Challenge, a yearly competition organized by the South Korean government intended to promote student involvement in open source software. In 2013, we are participating for the first time in the Google Summer of Code, a very competitive program that pairs students from around the world with mentors from different open source projects.

We have built an application on top of OMPL called OMPL.app, which adds integration with a collision checking library for checking the the validity of states and paths, a library for loading geometric meshes that describe robots and environments, and an easy-to-use GUI. The software comes with many example robot models and environments. Students are able to solve motion problems without writing any code through the GUI (shown in Fig. 2). A student can load a file that defines a motion planning problem, choose a planning algorithm, and simply click on the "Solve" button to obtain a solution. If a solution is found, it is played back by animating the robot along the found path. It is also possible to visualize a projection of the graph of valid states and valid transitions between states as constructed by the planning algorithm. This visualization makes it clear that different planning algorithms explore the search space in a qualitatively very different manner. It can also be used to gain some understanding of where a planning algorithm gets stuck. By default the GUI assumes the robot is a free flying rigid body, but this can be changed to give the robot the dynamics of a car, a quadrotor, or a blimp. In each case several algorithms can be selected and for each planner some parameters can be changed (although the default settings work well in most scenarios).

2 Motion Planning Project Design

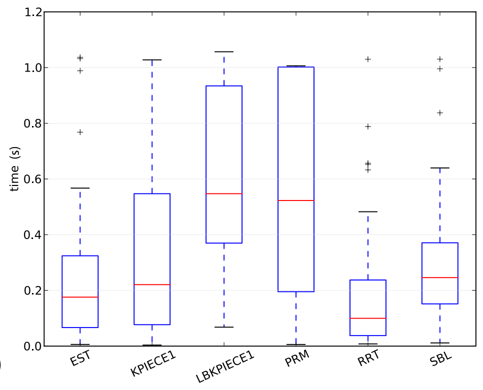
Course outcome criteria were formulated to evaluate whether students learned to model and solve realistic introductory and intermediate-level motion planning problems. Self-evaluative criteria were included to improve transfer of motion planning concepts and skills to more challenging problems. After completing the course students must be able to:

(1) Evaluate basic performance of several motion planning algorithms.
(2) Model motion planning problems.
(3) Solve motion planning problems with geometric constraints.
(4) Model and solve motion planning problems with differential constraints.
(5) Demonstrate good practices in software engineering when designing own algorithms.
(6) Demonstrate good practices in evaluating own algorithms.
(7) Highlight major results in graphs and figures.
(8) Write an organized and well-structured project report.

Based on these criteria we designed a series of four, increasingly difficult projects to introduce and then build the conceptual base, higher-order reasoning and technical skills needed to produce the motion planning solutions described below. Projects 1 and 2 are designed to teach outcomes 1–3, project 3 is used to teach outcomes criteria 4–6, and project 4 is used to teach outcomes 5–8. At Rice University all four modules are used, but it is also possible to use only selected projects in courses that need to cover a broader area of robotics. This was done at some of the partner institutions described in section 3.

Projects 1 and 2. The first two projects can be completed without any programming, yet instill a deep understanding of motion planning that is hard to convey through lectures alone. In the first project students are, among other things, asked to use the GUI to solve a number of motion planning problems with several different planners and make a qualitative assessment of what makes a motion planning problem "hard" and which planner is "best" at solving each problem. The motion planning problems are provided and are similar to the one shown in Fig. 2. The hardness of these problems is typically difficult to characterize and does not immediately follow from the theoretical complexity of motion planning. Sampling-based motion planning often uses randomization (and in the OMPL implementation all planners do), which also makes it harder to judge which algorithm is better at solving a particular problem.

Project 1 helps students understand that evaluating an algorithm's performance on specific examples is complex. This project prepares them for project 2 where the students are provided with a benchmarking tool to systematically compare performance of different algorithms. The benchmark runs planners several times on a given motion planning problem, collects a large number of performance characteristics, and aggregates the results in a summary report with bar graphs (for binary-valued variables), stacked bar graphs (for categorical performance variables) and box plots (for integer-valued and continuously-valued variables). Fig. 3 shows two example plots. The best planner can now be more precisely defined as, e.g., the planner with the lowest median, mean, or worst-case solve time. At the same time, this interpretation needs to be nuanced with the fact that some planners may not always find exact solutions (i.e., paths that reach the goal exactly), but are often able to find approximate solutions: paths that *almost* reach the goal. The same benchmarking framework can be used by students in later projects to evaluate their own algorithms.

(a)


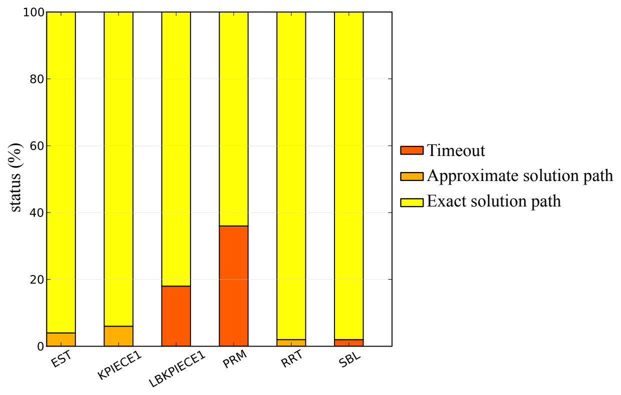
(b)


Figure 3. **Sample out of OMPL's benchmarking tool.** Both plots are based on 50 runs of each algorithm. **(a)** A box plot of run time for different algorithms whose names are indicated on the x-axis. **(b)** A stacked bar chart of planner status distributions. Other possible status values that did not occur in this case include *crash* and *invalid* start/goal state, which can provide useful diagnostic information.

Project 3. The third project is significantly more challenging and is typically done in pairs. It requires students to solve motion planning problems for systems that are described by ordinary differential equations (ODEs), specifically, a pendulum with bounded torque and a simple car-like robot. The equations of motion are provided in the form $\dot{q} = f(q, u)$ for these systems. The students are asked to write an function that implements f, create a state propagation function, create a state validation function, and use different planning algorithms to solve motion planning queries for the two systems. For the pendulum, the objective is to swing the pendulum up to a vertical position. This is challenging due to torque limits: the pendulum has to swing back and forth a number of times to gain enough momentum to swing up. The simplicity of the problem description on the one hand and the complexity of the solution helps students appreciate the power of the underlying planning algorithms. The answer cannot easily be guessed or derived by hand. Solving the car-like system is done analogously. The main difference is that it includes a few obstacles that the car needs to avoid, which translates to a bit of extra code in the state validity checker. In the initial version of the assignment students were asked to implement their own numerical integration, but this turned out to be more time-consuming for students than anticipated. As a result, we added ODE solvers (i.e., numerical integration routines) to OMPL as well as convenience functions that turn an ODE function into a state propagator. After the students solve these problems with the built-in planning algorithms of OMPL, they are asked to implement a variant of one of those motion planning algorithms.

Project 4. The fourth and final project is typically also done in pairs. Students can choose one from a list of projects. They cover various advanced topics in motion planning such as path clustering, path optimization, planning for a robot arm with second-order dynamics, a comparison of centralized and decentralized multi-robot planning, elastic kinematic chains, and anytime planning. While some projects have been part of the teaching module since the beginning, we keep adding new projects, some of which are based on very recent conference and journal papers. This reinforces the relevance of what the students have learned in the class to real robotics applications. At this point students should be able to understand these papers, implement algorithms described in them, and evaluate their algorithm's performance. As part of the project, students are asked to write a report and give a presentation.

To make the material more accessible to students with diverse backgrounds, we recommend including a "warm-up" project to ease the transition between projects 2 and 3. At Rice we used an ungraded project between project 2 and 3 that prepares them for writing code using OMPL. In this project, students are asked to implement a new planner that randomly explores the search space and compare this to existing planning algorithms in the OMPL library. Random exploration does not make for a particularly good planning strategy, but it prepares students for projects

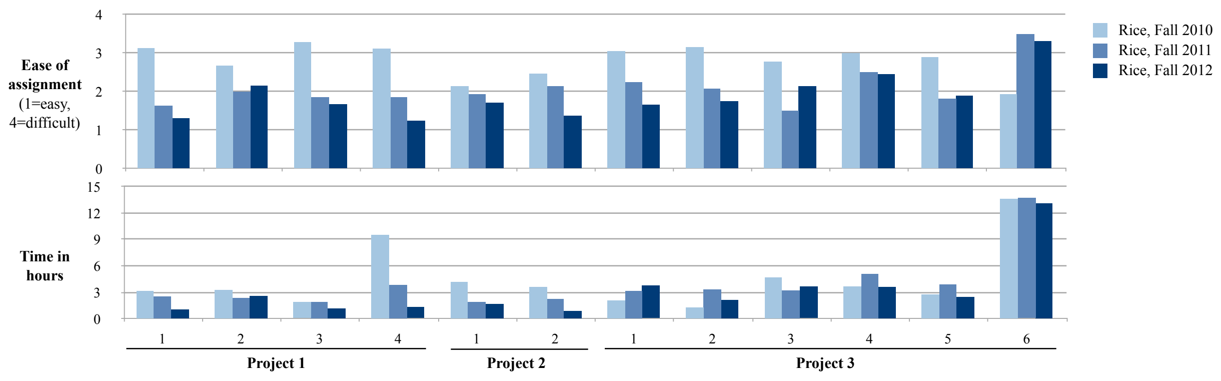


Figure 4. Reported ease of use and time spent on components of projects 1, 2, and 3 at Rice University.

3 and 4 as follows. First, it encourages the students to become familiar with the code and with navigating the documentation. Second, it exposes the students to the benchmarking facilities at the code level. Last but not least, students learn that the heuristics employed by the planning algorithms covered in class and provided by OMPL result in significantly improved performance over the "naïve" random exploration, even though the algorithms in OMPL also rely on random sampling.

3 Assessment

3.1 *Formative Assessment*

Over the last three years, a usability survey has been conducted on an ongoing basis throughout the course to obtain student feedback that has helped us adjust the degree of challenge projects pose for undergraduates in an intermediate level course and to improve the technology and its documentation. As a result OMPL students can complete significantly more projects of greater challenge than before the technology and the current projects were introduced. Before, students could not implement a single sampling-based planner in a semester. This was due in part to the amount of programming involved but mainly to the sophistication of the programming required for the correct implementation of advanced algorithmic concepts. Moreover, since students struggled all semester to implement one planner, that left no time to learn how various factors affected algorithmic performance. Now that the technology provides a class hierarchy to work with and students no longer need to implement many of the low-level data structures, they can explore, compare and evaluate different algorithms. Basic algorithmic performance is now explored in the first two projects, and students can evaluate performance for multiple sampling-based planners. In addition, despite the increased difficulty of the work, 81% of the students report that they would recommend the class to their peers (79% (15/19) in the 2010 offering of the class and 85% (11/13) in 2011).

Rice students voluntarily provided feedback after each project during the first three years of the project. The number of students participating over three years ranged from 14 to 21 students a year. Student surveys were also collected at the Worcester Polytechnic Institute (WPI; 18 participants) and the National University of Singapore (NUS; 16 participants) this past year. At Rice University a majority of the students were undergraduates, while at WPI and NUS undergraduates formed about a quarter of the students. At the US institutions (i.e., Rice and WPI) 22% of students were female and 8% of students came from underrepresented minorities. While at Rice University most students were CS majors, the students at WPI and NUS had a more diverse background in engineering disciplines. As a result the students at WPI and NUS may have been less knowledgeable about algorithms and less experienced in programming. Two measures of effort—the ease of completing project components and the amount of time they spent on each component—indicate that Rice students perceive the revised projects as easier to complete

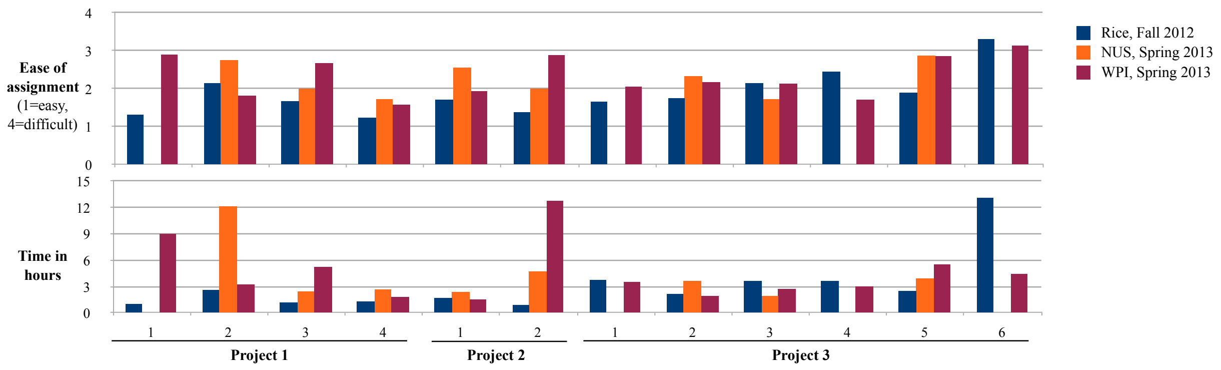


Figure 5. Ease of use and time spent on components of projects 1, 2, and 3 at three schools.

and requiring less time with software and project improvements over the years (see Fig. 4). At WPI a few components were optional (e.g., part 4 of project 3). Most likely only the stronger students opted to do those, so we would expect these components to be perceived as relatively easy to complete. Also, at NUS some project components were omitted by the instructor, thus no measures are available for them. Still, WPI and to some extent NUS students reported that a few components were more difficult to complete and required more time than Rice students (see Fig. 5). Students at WPI and NUS also reported more issues with software installation of the software and the documentation of a few specific topics. At Rice University, these issues were mitigated by direct access to OMPL developers as well as teaching assistants who were intimately familiar with OMPL. Steps will be taken this summer to improve OMPL's ease of installation and documentation. The numbers in Figures 4 and 5 refer to the component numbers in the list below.

Project 1 (1) Getting started on a computer cluster with OMPL pre-installed, (2) downloading and installing OMPL on student's own computer, (3) creating an environment to compile programs that link against OMPL, and (4) using different planning algorithms in the GUI for different motion planning problems.

Project 2 (1) Running the benchmark program, and (2) modifying the sampling strategy used by a motion planning algorithm.

Project 3 Understanding the (1) pendulum system and (2) car-like system, use the provided (3) RRT and (4) KPIECE planners for these systems, (5) visualize the solution paths, and (6) modify the RRT planner to create the Reachability-Guided RRT planner.

3.2 *Can students produce competent solutions to motion planning problems?*

We created an analytic rubric to assess whether students completing the course acquired the conceptual base, higher-order reasoning, and technical skill needed to solve introductory motion planning problems (see Table 1). Performance achievements were identified that specifically defined and discriminated each of three broad levels of competency (Developing, Competent, Accomplished) for each of the outcome criteria. A 6-point scale was used to further discriminate performance within a level (0 = Little or no knowledge or skill, 1 = Developing competence, 2 = Competent, 3 = Highly competent, 4 = Accomplished, 5 = Highly accomplished). The rubric contains very specific information to help raters discriminate performance levels. Projects 1–4 from the Rice fall 2012 semester were scored using this rubric by the course's graduate teaching assistant, a Ph.D. student working on the topic of motion planning. See Fig. 6 for the results from this summative assessment. 90% or more of the students exhibited competent or exemplary achievement levels on 6 of the 8 outcomes. Specifically, 100% of the students could evaluate basic performance of motion planning algorithms, 94% could model motion planning problems, 100% could solve motion planning problems with geometric constraints, 95% could demonstrate good practices in evaluating their own algorithms, 100% could highlight major

Table 1. Rubric for assessing motion planning knowledge and skills.

Outcome Criteria	**Levels of Achievement**		
	Developing	**Competent**	**Accomplished**
Students can: **1.** Evaluate basic performance of motion planning algorithms	• Was able to run several planners on several problems and create plots of performance metrics	• Was able to run several planners on several problems and create plots of performance metrics • Compared performance across different planning instances and with different samplers • Provided plausible reasons for the performance of planners based on acquired results	• Was able to run several planners on several problems and create plots of performance metrics • Compared performance across different planning instances and with different samplers • Provided plausible reasons for the performance of planners based on acquired results • Was able to synthesize and summarize results and explain conditional nature of performance
2. Model motion planning problems	• Slight modeling errors, but solved motion planning problems	• Modeled configuration spaces and state validity checkers correctly • Reasonable implementation of model	• Modeled configuration spaces and state validity checkers correctly • Simple, but no simpler than needed, implementation of model
3. Solve motion planning problems with geometric constraints	• Can solve geometric motion planning problems with at least one planner	• Can solve geometric motion planning problems with several planners	• Can solve geometric motion planning problems with several planners • Changed planner parameters to improve performance
4. Model and solve motion planning problems with differential constraints	• Mostly correct implementation of state propagation function • Created and used own numerical integration routines or loosely used OMPL-provided ODE solvers • Obtained very approximate solutions to motion planning problems	• Correct implementation of propagation • Correct use of an OMPL-provided ODE solver • Obtained close to exact solutions to motion planning problems	• Correct implementation of propagation • Correct use of an OMPL-provided ODE solver • Obtained close to exact solutions to motion planning problems • Changed planner parameters to improve performance
5. Demonstrate good practices in software engineering when designing own algorithms	• Submitted code that compiles and runs	• Submitted code that compiles and runs • Code documentation includes input/output and correct description of functions	• Submitted code that compiles and runs • Code was well-documented • Classes and files were conceptually well-organized
6. Demonstrate good practices in evaluating own algorithms	• Chooses an appropriate performance metric	• Chooses one or more appropriate performance metrics • Evaluates performance using a number of different parameter settings	• Chooses one or more appropriate performance metrics • Evaluates performance using a number of different parameter settings • Interprets the results
7. Highlight major results in graphs and figures	• Used appropriate type of illustration or chart for data • Legible labels, titles, legend and chart or figure elements • Reasonable content in labels, titles and legend	• Used appropriate type of illustration or chart for data • Legible labels, titles, legend and chart elements • Reasonable content in labels, titles and legend • Graphs show relationships and significant results	• Used appropriate type of illustration or chart for data • Legible labels, titles, legend and chart elements • Strong content in labels, titles and legend • Graphs emphasize relationships and significant results
8. Write organized and well-structured project reports	• One or more parts of the report are missing: problem description, explanation of solution, results, and interpretation of results	• Report contains enough information about the problem, solution, results, and discussion of results so that another student who takes the class can understand the report	• Report contains enough information about the problem, solution, results, and discussion of results so that another student who takes the class can understand the report • Report is distinguished by one or more: effective descriptions, complete and well-justified explanations, clear results, and strong concluding arguments in the discussion

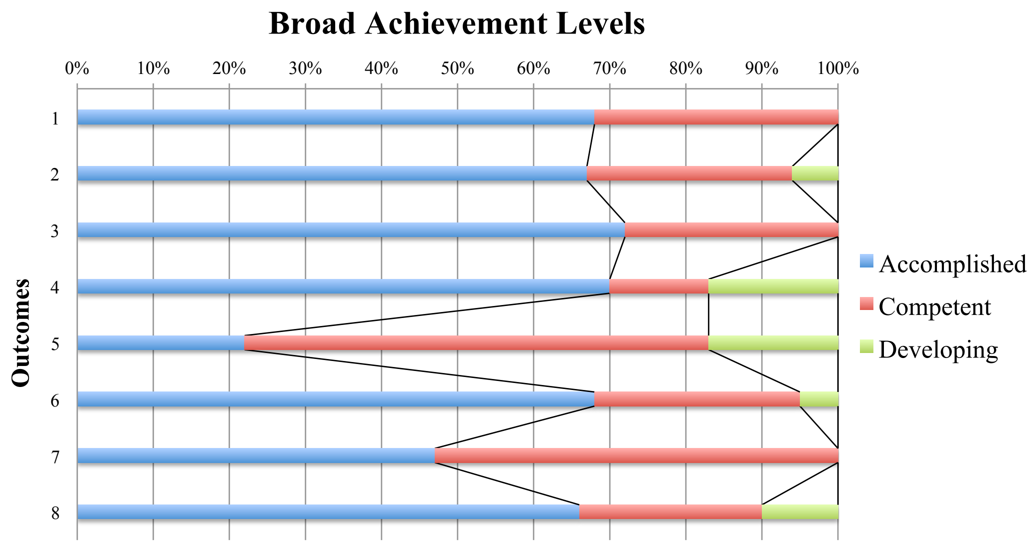


Figure 6. Competency of students at Rice during the Fall 2012 semester.

results in graphs and figures, and 90% could write organized and well-structured project reports. However, two outcomes were more challenging. Only 83% of the students could model and solve motion planning problems with differential constrains or demonstrate good practices in software engineering when designing their own algorithms.

4 A Bridge to Research and Industrial Applications

In addition to helping students achieve the course's short-term learning outcomes, the project-based approach to the algorithmic robotics curriculum appears to be inspiring undergraduates to choose research careers in robotics. The last teaching module (Project 4) introduces students to topics in motion planning in which researchers are still producing new discoveries. At Rice University, where over 60% of the undergraduates participate in research projects, it is not uncommon for students in the course to become so excited about motion-planning research that they subsequently complete a motion planning research project for course credit. Since this course allows them to acquire enough depth in the specific modeling and good practices motion planning requires using the same technology used by researchers, they are well-prepared to begin research in the field. By then completing research projects students have the opportunity to acquire theoretical knowledge as well as more practical software engineering skills. For example, they learn how to work with several others on a large code base using a distributed version control system, and they learn good practices for testing and documenting code. As a result, these undergraduate projects are often a stepping stone to graduate school. Several undergraduates who have taken the class at Rice have gone to graduate school to study robotics.

Independently of work done in our research group, a growing number of graduate students conducting research in motion planning are using OMPL, including several doctoral students and post-doctoral fellows, who have published papers using OMPL (Marble & Bekris, 2013; Barry et al., 2013). In short, the use of OMPL is now growing rapidly in the robotics community outside of our research group. Moreover, robotics researchers have started to contribute OMPL-based implementations of new algorithms for inclusion in future releases.

Part of OMPL's appeal to researchers is that it is has been integrated with the widely-used Robot Operating System (ROS) (Quigley et al., 2009), an Open Source software environment that runs on top of Linux (and to some extent on OS X and MS Windows). Within ROS OMPL is used to, e.g., plan motions for the arms of the PR2, a mobile manipulator with 7 joints in each arm (see Fig. 7). Our PBL teaching module thus exposes students to software that has

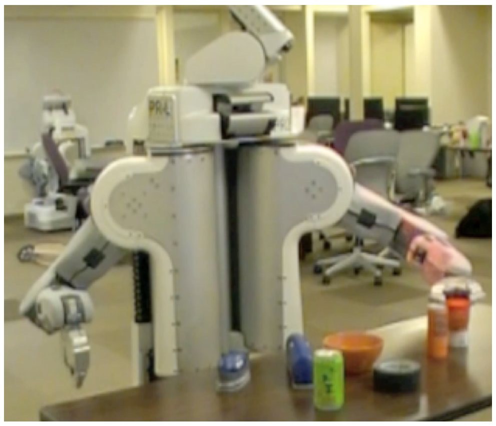
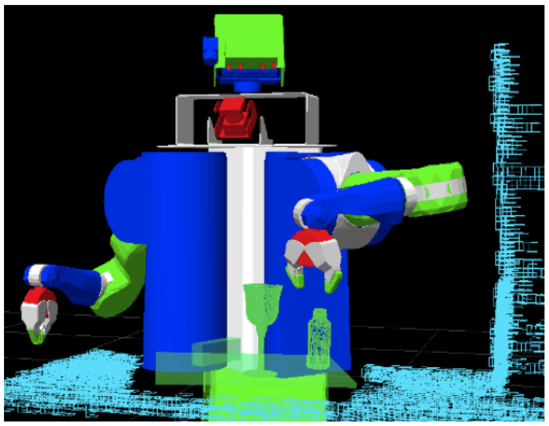
(a)  (b) 

Figure 7. **(a)** A PR2 robot from Willow Garage performing manipulation tasks using OMPL. **(b)** The sensed model of the world used for planning.

been adopted by the robotics community for research purposes. Knowing that the same software can be used for difficult real-world problems is motivating for the students. Since many robot hardware platforms use the ROS system, any new motion planning algorithm that is added to OMPL that conforms to the abstract planner API can immediately be used on a wide variety of hardware platforms.

5 Discussion

We have described our software for Project-Based Learning of robot motion planning. We have developed a teaching module consisting of a number of projects that scaffold learning of key concepts in robot motion planning. The module is currently targeted at advanced undergraduate students and graduate students, but our long-term goal is to make the module more accessible earlier in a CS curriculum so that student are inspired and prepared to succeed in research careers in robotics. Selected parts of the teaching module may eventually also be used in an algorithm class to illustrate search algorithms in high-dimensional spaces, randomized algorithms, and computational geometry algorithms for, e.g., collision checking and nearest-neighbor queries.

Our experience so far indicates that it is possible to create software for PBL that is accessible to diverse learners, yet still powerful enough to run on real robots. This software is of great interest to the robotics community, which is actively sustaining it. In future work we will explore the extent that real robot hardware and physically realistic robot models can be made part of our teaching module. Recently released software (Chitta et al., 2012) that builds on OMPL and the rest of the ROS software ecosystem has made it significantly easier to apply OMPL's algorithms to both real and simulated robots of very diverse types. At institutions that offer several robotics classes a common hardware platform and software infrastructure, which can be used for motion planning, computer vision, control, and other related robotics areas can provide a compelling environment for an integrated robotics education.

6 Acknowledgments

This project is funded by NSF CCLI 0920721 and by Willow Garage. The authors are indebted to Ioan Şucan for his significant role in the development of OMPL and to other members of the Kavraki Lab for their contributions to OMPL and other motion planning software that preceded it and provided some inspiration for the current design. We are grateful to the faculty members and students at NUS and WPI for their participation in our project.

References

Balch, T., Summet, J., Blank, D., Kumar, D., Guzdial, M., O'Hara, K., et al. (2008). Designing Personal Robots for Education: Hardware, Software, and Curriculum. *IEEE Pervasive Computing*, *7*(2), 5–9.

Barron, B.J.S., Schwartz, D.L., Vye, N.J., Moore, A., Petrosino, A., Zech, L., et al. (1998). Doing with understanding: Lessons from research on problem-and project-based learning. *Journal of the Learning Sciences*, *7*(3-4), 271–311.

Barry, J., Kaelbling, L.P., & Lozano-Perez, T. (2013). A Hierarchical Approach to Manipulation with Diverse Actions. In IEEE Intl. Conf. on Robotics and Automation (pp. 1791–1798).

Canny, J. (1988). *The Complexity of Robot Motion Planning.* Cambridge, MA: MIT Press.

Chitta, S., Şucan, I., & Cousins, S. (2012). MoveIt!. *IEEE Robotics & Automation Magazine*, *19*(1), 18–19.

Choset, H., Lynch, K.M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L.E., et al. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementations.* MIT Press.

Chung, J.C.C., & Chow, S.M.K. (2004). Promoting student learning through a student-centred problem-based learning subject curriculum. *Innovations in Education and Teaching International*, *41*(2), 157–168.

Şucan, I.A., & Kavraki, L.E. (2012). A Sampling-Based Tree Planner for Systems With Complex Dynamics. *IEEE Trans. on Robotics*, *28*(1), 116–131.

Şucan, I.A., Moll, M., & Kavraki, L.E. (2012). The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, *19*(4), 72–82. http://ompl.kavrakilab.org

Galand, B., & Frenay, M. (2005). *L'approche par problèmes et par projets dans l'enseignement supérieur: Impact, enjeux et défis.* Presses Universitaires de Louvain.

Gijbels, D., Dochy, F., Van den Bossche, P., & Segers, M. (2005). Effects of problem-based learning: A meta-analysis from the angle of assessment. *Review of educational research*, *75*(1), 27–61.

Graham, R., & Crawley, E. (2010). Making projects work: a review of transferable best practice approaches to engineering project-based learning in the UK. *Engineering Education*, *5*(2), 41–49.

Kay, J., Barg, M., Fekete, A., Greening, T., Hollands, O., Kingston, J.H., et al. (2000). Problem-Based Learning for Foundation Computer Science Courses. *Computer Science Education*, *10*(2), 109–128.

Latombe, J.C. (1991). *Robot Motion Planning.* Boston, MA: Kluwer Academic Publishers.

LaValle, S.M. (2006). *Planning Algorithms.* Cambridge University Press.

Marble, J., & Bekris, K.E. (2013). Asymptotically Near-Optimal Planning with Probabilistic Roadmap Spanners. *IEEE Transactions on Robotics*, *29*, 432–444.

Prince, M.J., & Felder, R.M. (2006). Inductive teaching and learning methods: Definitions, comparisons, and research bases. *Journal of Engineering Education*, *95*(2), 123–138.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., et al. (2009). ROS: an open-source Robot Operating System. In ICRA Workshop on Open Source Software .

Touretzky, D.S. (2010). Preparing computer science students for the robotics revolution. *Commun. ACM*, *53*(8), 27–29.

Tsianos, K.I., Halperin, D., Kavraki, L.E., Latombe, J.C., & Attalah, M. (2008). Robot Algorithms. *Algorithms and Theory of Computation Handbook* CRC Press.