Teaching Motion Planning Concepts to Undergraduate Students

Mark Moll, *Member, IEEE,* Ioan A. Şucan, *Student Member, IEEE,*
Janice Bordeaux, and Lydia E. Kavraki, *Senior Member, IEEE*

***Abstract*—Motion planning is a central problem in robotics. Although it is an engaging topic for undergraduate students, it is difficult to teach, and as a result, the material is often only covered at an abstract level. Deep learning could be achieved by having students implement and test different algorithms. However, there is usually no time within a single class to have students completely implement several motion planning algorithms as they require the development of many lower-level data structures. We present an ongoing project to develop a teaching module for robotic motion planning centered around an integrated software environment. The module can be taught early in the undergraduate curriculum, after students have taken an introductory programming class.**

I. Introduction

Robots play an increasingly important role in our daily lives. For many familiar applications such as urban search-and-rescue, planetary exploration, and household chores it is critical that robots efficiently find paths (i.e., motion plans) from starting locations to desired goals. Although motion planning at an abstract level is a fairly intuitive concept, it is a topic that is difficult to teach. The difficulty arises because deep understanding of motion planning concepts requires hands-on experience with different algorithms. However, the low-level programming details in the implementation of these algorithms can often obscure high-level concepts, which are necessary for a solid understanding of the subject. This paper describes an ongoing project that aims to provide the software tools and curricular material that will help fill some of the gaps in undergraduate robotics education. Many colleges offer introductory robotics classes early in the curriculum, often to attract students to Science, Technology, Engineering, and Mathematics (STEM) disciplines. However, any subsequent robotics classes are often only available at the senior undergraduate level or beginning graduate level. They tend to include many topics in an effort to teach the students as much as possible about an extremely diverse area. Available robotics textbooks [1–3] cover many topics and challenge the student as well as the instructor to keep up.

The project described herein focuses exclusively on one core concept in robotics: motion planning. Motion planning is a mature field within robotics and often comprises a significant part of advanced robotics classes. In two recent robotics textbooks, motion planning is almost the sole focus of one [3], while taking up more than half of the other [1]. Within our project, we have developed software and teaching materials to make motion planning more accessible to diverse learners earlier in the undergraduate curriculum. We focus exclusively on sampling-based planners [1, Chapter 7]. At the same time, we aim to make it easier for educators to teach the material. We have identified instructors at other institutions who have agreed to use the material in their class, and through a feedback loop we will iteratively refine and extend the existing material. The existing material is also freely available to anyone else for use in their classes as they see fit.

Motion planning has been studied extensively in the past. The basic problem of finding collision-free paths for polyhedral robots operating in a polyhedral workspace is already PSPACE-complete [4]. Moreover, 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. These algorithms are extremely popular, as they have shown good experimental performance, but can be hard to implement correctly [5]. This is why our software is currently focused on those algorithms. Sampling-based algorithms often provide probabilistic completeness: they will find a solution with probability 1 if one exists, but cannot decide if no solution exists. The fundamental idea of sampling-based motion planning is to approximate the connectivity of the 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. There is a wide variety of sampling-based algorithms, but they almost all share common components. Below we briefly describe the most important ones:

State space Points in the state space (or configuration 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.

Control space Points in the control space represent inputs that can be applied to change the state of the system. This is only needed for systems with dynamics. For purely geometric planning no controls are needed and state interpolation is used instead to compute path segments.

Sampler Samplers are needed to generate different states from the state space. For control-based planning, another

M. Moll, I. Şucan, and L. Kavraki are with the Department of Computer Science at Rice University, Houston, TX 77005, USA. Email: {isucan,mmoll,kavraki}@rice.edu. J. Bordeaux is with the George R. Brown School of Engineering, Rice University, Houston, TX 77005, USA. Email: jbordeau@rice.edu

sampler is needed to generate different controls.

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 For geometric planning the local planner usually performs some kind of interpolation in the state space between two different states. For planning with controls the local planner is a function that computes a resulting trajectory when a control is applied for some period of time starting from some initial state.

The motion planning curriculum we have developed enhances deep understanding of these concepts as well as several state of the art motion planning algorithms. The curriculum is supported by an integrated software environment described in the following section.

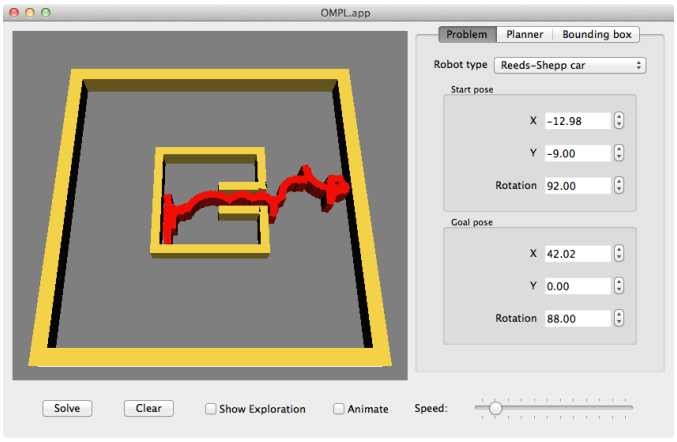

Fig. 1. The OMPL.app graphical interface. A solution path is shown for a car-like robot driving out of a "bug trap" environment.

II. Motion Planning Software Development

A. The Open Motion Planning Library

The software we have developed is based on the Open Motion Planning Library [6], which we released in December 2010. This C++ library was designed to have a clear mapping between the concepts described above and C++ classes. We have created Python bindings for the library, allowing students to access almost all of OMPL's functionality through Python as well. Since many institutions use Python for introductory programming classes, the Python bindings are intended to help in making the code more accessible to novice programmers. The OMPL library is also efficient, lightweight, easy to integrate with external software, and easy to install. It contains implementations of many sampling-based algorithms such as PRM [7], RRT [8], KPIECE [9], SBL [10], EST [11], and many more.

B. OMPL.app: A Graphical User Interface for OMPL

We have developed an additional software layer called OMPL.app, based on OMPL. OMPL.app adds integration with PQP [12], a collision checking library, and Assimp [13], a library for reading CAD files, and defines a state validity checker for 2D and 3D rigid body motions that checks whether states are collision-free. Furthermore, OMPL.app includes many example models for robots and environments, so that it is easy for students to get started. Students are able to solve motion problems without writing any code through a GUI that is also included (shown in Fig. 1). A student can load meshes that represent the environment and a robot, define start and goal states, 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. Optionally, a trace of the found path can replace the animation (by unchecking the "Animate" check-box). It is also possible to show a workspace projection of the states that were explored by the planner, which can be helpful in tuning planner parameters or with selecting the appropriate planning algorithm for a particular problem. By default, the program assumes that students want to plan for a free-flying 3D rigid body (i.e., the state space is SE(3)), but planning in SE(2) is possible as well. We have also pre-defined a number of common robot types that require controls: a blimp, a quadrotor, and a number of car models. For each robot type, the appropriate planners can be selected in the "Planner" tab (otherwise a default one will be automatically selected). Once a planner is chosen, its parameters can be tuned, if desired. Finally, the student can adjust the bounding box for the robot's position. By default this is the bounding box of the environment mesh.

C. An Example Program

OMPL offers a simple API to maximize ease of use. Figure 2 shows the complete code necessary for planning the motion of a rigid body between two states, in Python and C++. In both cases, the only steps taken in the code are: instantiate the space to plan in (SE(3), line 1), create a simple planning context (using SimpleSetup, line 4), specify a function that distinguishes valid states (line 6), specify the input start and goal states (lines 8–13), and finally, compute the solution (line 14). The SimpleSetup class initializes instantiations of the core motion planning classes with reasonable defaults, which can be overridden if desired.

Essentially, the execution of the code can be reduced to three simple steps: (1) specify the space in which planning is to be performed, (2) specify what constitutes a valid state, and (3) specify the input start and goal states. Such simple specifications are desirable for many users who simply want motion planning to work, without having to select problem specific parameters, or different sampling strategies, different planners, etc. The simplicity of the approach we provide is especially useful for students as it allows them to learn incrementally and still solve problems, without having to input a large number of parameters from the beginning. This capability is made possible by OMPL's automatic computation of planning parameters. In the example above, a planner is automatically selected based on the specification of the goal and the space to plan in. The selected planner is then automatically configured by computing reasonable default settings that depend on the planning context. If a user decides to choose their own planner,

```python
space = SE3StateSpace()
# set the bounds (code omitted)

ss = SimpleSetup(space)
# "isStateValid" is a user-supplied function
ss.setStateValidityChecker(isStateValid)

start = State(space)
goal = State(space)
# set the start & goal states to some values
# (code omitted)

ss.setStartAndGoalStates(start, goal)
solved = ss.solve(1.0)
if solved:
    print setup.getSolutionPath()
```

```cpp
StateSpacePtr space(new SE3StateSpace());
// set the bounds (code omitted)

SimpleSetup ss(space);
// "isStateValid" is a user-supplied function
ss.setStateValidityChecker(isStateValid);

ScopedState<SE3StateSpace> start(space);
ScopedState<SE3StateSpace> goal(space);
// set the start & goal states to some values
// (code omitted)

ss.setStartAndGoalStates(start, goal);
bool solved = ss.solve(1.0);
if (solved)
    setup.getSolutionPath().print(std::cout);
```

Fig. 2. Solving a motion planning problem with OMPL in Python (left) and in C++ (right).

or set their own parameters, OMPL allows the user to do so completely—no parameters are hidden.

III. Motion Planning Curriculum Development

We have developed a series of assignments that complement the material presented in class. The assignments are set up so that each assignment builds on the previous one, but without directly depending on it. Furthermore, for each assignment we have developed several versions that vary in depth. One of the aims of our project is to make the material accessible to a broad audience of engineering students, early in the curriculum. In terms of required programming skills, we believe the assignments we have developed can be completed if students have taken an introductory programming class in Python or C++. Since all of the components common to all motion planning algorithms are already implemented, students do not have to spend time implementing any of the low-level data structures, thereby allowing them to focus on the high-level algorithms.

A. Evaluating Motion Planning Algorithms

For students to develop an intuition about the complexity of motion planning, it is useful for them to apply different algorithms in OMPL to a variety of example problems. We have created a number of scenarios that can be used with the OMPL.app GUI. Students are asked to run different planners on different problems, experiment with different motion planning algorithms, visualize the output, and reflect on the reported solutions. In a second assignment this process is more formalized through a simple programming assignment where students benchmark motion planning algorithms and different sampling strategies. To aid with this assignment, a Benchmark class is provided. This class automatically collects a large number of statistics, and a post-processing script can be used to analyze the statistics and automatically produce plots for measured data (see Fig. 3). The objective of the assignment is for students to understand performance measures and the effect of certain parameters on performance.

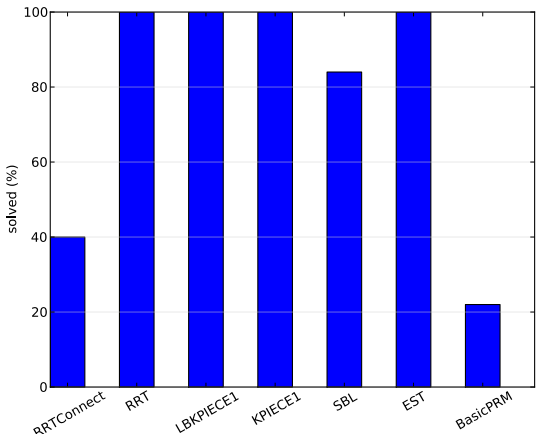

Fig. 3. The Benchmark makes it easy to automatically generate plots that compare performance metrics such as success rate across different planning algorithms.

B. Implementing a New Planning Algorithm

After students are familiar with the organization of OMPL and know how to run a planning algorithm, they are asked to implement a variant of a planning algorithm that is already included in OMPL (e.g., implement ADDRRT[14], given the version of RRT in OMPL). This will encourage deep learning of the material without requiring excessive programming. The students also need to apply their newly added algorithm to more challenging problems.

C. Open-ended Projects

For the final assignment, the students can choose from a number of assignments designed around advanced topics in motion planning. Although specific goals are given in each assignment, students are encouraged to research the problem and come up with creative solutions. We have created assignments on topics such as:

Path clustering The type of algorithms implemented in OMPL return a different path on each run. In this assignment students are asked to cluster multiple solution paths for the same motion problem into clusters of similar paths. Such clusters often correspond to a homotopy class, a concept from topology.

Path optimization There are a variety of techniques available that take the output of a motion planning algorithm and attempt to make make the path smoother, shorter, etc. This assignment explores some of these techniques.

Dynamic manipulation In this assignment students learn how to plan for a n-link planar manipulator whose dynamics are provided in the form of a system of differential equations. The students are asked to explore how the complexity of the problem changes as velocity/control bounds are changed, links are added, or if some links are under-actuated.

Multi-robot planning The algorithms in OMPL can be used to plan motions for a group of robots. However, this becomes very challenging if the individual robots need to carefully coordinate their actions to reach their respective goal positions.

IV. Assessment

To evaluate the effectiveness of the technical and curricular activities, we are conducting a formative and summative formal outcomes assessment. We have identified a number of partners at other educational institutions who plan to use the material in their own classes. The learning outcomes will be monitored periodically through a series of surveys conducted in a robotics class at Rice and our partner institutions. We plan to work closely with the partner institutions, so that feedback can be used to quickly improve the curriculum materials and teaching methods. The success of the project is determined by evaluating the overall level of achievement attained by students.

V. Dissemination

One of our objectives is to broaden the dialog between students and faculty and beyond institutional boundaries to create a worldwide community of OMPL users. The software is distributed under a BSD license, allowing anyone to modify the software as needed, and is also available through a public repository, so that serious OMPL users can track the development and contribute their own extensions. Additionally, there is an active mailing list of OMPL users and developers. We are also collaborating with Willow Garage on the integration of OMPL into ROS, a popular robotics software system that runs on many robots, and is also used in other robotics classes. The technical skills that students acquire with OMPL can thus be leveraged in such classes.

OMPL comes with a API reference manual, but we have also created many demo programs and tutorials which (1) teach students how to get started with OMPL, (2) illustrate common workflows, and (3) demonstrate usage patterns for some of the more advanced features. We are currently also preparing a comprehensive tutorial, which will include sets of slides and a handout with exercises. The tutorial will be held first at a robotics conference (the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2011) and is thus aimed at a more advanced audience than the typical undergraduate student. Nevertheless, the tutorial will be valuable in two ways. First, many attendees will be educators themselves, and they will be better prepared to teach the material after the tutorial. Second, many parts of the tutorial will still be accessible to undergraduate students, and—with little effort—can be used to tie together some of the existing mini tutorials.

VI. Conclusion

We have described a teaching module that can be used in an undergraduate robotics class to teach robot motion planning. It enhances deep learning and is supported by an integrated software environment. The assignments we have developed scaffold learning, avoid heavy programming, and focus on critical thinking with a hands-on approach. As the project advances, the products of this project can strengthen other courses in computer science and engineering, and assist in retaining students in STEM disciplines.

Acknowledgements

This project is funded by NSF CCLI 0920721, NSF IIS 0713623, and by Willow Garage. The authors are indebted 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.

References

[1] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[2] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.

[3] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006. URL http://msl.cs.uiuc.edu/planning/.

[4] J. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1988.

[5] I. A. Şucan and L. E. Kavraki. On the implementation of single-query sampling-based motion planners. In *IEEE Intl. Conf. on Robotics and Automation*, pages 2005–2011, may 2010. doi:10.1109/ROBOT.2010.5509172.

[6] I. A. Şucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library, 2010. URL http://ompl.kavrakilab.org.

[7] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. on Robotics and Automation*, 12(4):566–580, August 1996. doi:10.1109/70.508439.

[8] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *Intl. J. of Robotics Research*, 20(5):378–400, May 2001. doi:10.1177/02783640122067453.

[9] I. A. Şucan and L. E. Kavraki. A sampling-based tree planner for systems with complex dynamics. *IEEE Trans. on Robotics*, 2011. doi:10.1109/TRO.2011.2160466.

[10] G. Sánchez and J.-C. Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In *The Tenth International Symposium on Robotics Research*, pages 403–417, 2001. doi:3-540-36460-9_27.

[11] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *Intl. J. of Computational Geometry and Applications*, 9(4-5):495–512, 1999.

[12] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha. Fast distance queries with rectangular swept sphere volumes. In *IEEE Intl. Conf. on Robotics and Automation*, pages 3719–3726, 2000. doi:10.1109/ROBOT.2000.845311.

[13] Assimp, open asset import library. http://assimp.sf.net, 2011 .

[14] L. Jaillet, A. Yershova, S. M. LaValle, and T. Siméon. Adaptive tuning of the sampling domain for dynamic-domain RRTs. In *IEEE International Conference on Intelligent Robots and Systems*, 2005.