Programming assignments to learn how to build a Probabilistic Roadmap Planner

Jan Rosell, Alexander Pérez.

IOC- Divisió de Robòtica

IOC-DT-P-2008-11
Setembre 2008

Programming assignments to learn how to build a Probabilistic Roadmap Planner

Jan Rosell and Alexander Pérez

Abstract

Research in robot motion planning requires the availability of a simulation environment where to test and validate the theoretic contributions. Nevertheless, PhD programs and graduate studies that include robot motion planning courses usually cover the subject only from a theoretical perspective. Therefore, students aiming to focus their research in this line, face the big challenge of developing their own simulation environment. To cope with this problem, this paper proposes the use of a programming framework based on multi-platform open source code, and presents a set of twelve programming assignments to allow students of any robot motion planning course a quick mastering of the basic skills involved, covering from graphical rendering issues to collision detection or graph representations and algorithms.

I. INTRODUCTION

PhD. programs and graduate studies in robotics usually include motion planning among its main courses. These courses should include some basic robot simulation issues like the modeling of objects, collision detection and other computational geometry problems, graph representations and algorithms, and graphic rendering and other graphical user interface issues. Their teaching is necessary to fill the gap between theory and practice in robot motion planning courses. The mastering of these issues can help the students that later focus their research interest in motion planning to program their simulation environment where to test and validate their theoretic contributions.

There are some open source motion planners available on the web like *An Object-Oriented Programming System for Motion Planning* (OOPSMP, www.kavrakilab.org/software), the *Motion Planning Kit* (http://ai.stanford.edu/mitul/mpk) or the *Motion Strategy Library* (http://msl.cs.uiuc.edu/msl/). Although they have interesting features, they do not cover the needs from a teaching perspective. They are not easy to use and it is hard to get an insight into their structure in order to adapt and extend them to the particular needs. This is the reason why, in the scope of robot motion planning courses, the inclusion of robot simulation issues is interesting, since it gives the students the opportunity to face different related problems and to acquire a global view of the field.

The aim of this paper is first to propose a programming framework that cover most of the robot simulation issues needed for the development of motion planning software. Having in mind an open source philosophy and a multi-platform approach, Section II presents the set of programming tools of this framework. The second aim is the proposal of a method to easily acquire the mastering of all these tools. Section III presents a set of twelve programming assignments for a robot motion planing course that, as building bricks, incrementally present all the key aspects. Starting from a simple program that models basic geometries, it ends up with a basic probabilistic roadmap planner for free-flying robots. Finally, Section IV concludes the work.

II. TOOLS

A. Software development tools

The development of a motion planning application in robotics requires of the use of software development tools to correctly manage it [1]. This includes the necessity to control versions for tracking and debugging purposes, and to easily generate a good documentation. Also, for a wide spreading and ease of use, it is interesting to develop a versatile application able to run in different platforms and that do only require open software libraries.

Programming paradigms and languages: One of the most published programming paradigms is the Object Oriented Programming (OOP, [2]), since it allows to comfortably work with the models (objects) of all the actors involved in the problem, like the Configuration Space, the Work Space, the Obstacles and the Robots for the case of motion planning problems. OOP allows the creation of modular and reusable code. One step further lies the Generic Programming paradigm that complements OOP enlarging its flexibility. The development of software following the OOP paradigm can be backed by the use of the Unified Modeling Language, a meta language used to graphically show a software design [3]. Using an UML graphic tool like StarUML (www.staruml.com) it is possible to make an initial global design at class level, that can later be refined or modified.

A widespread choice for the programming language is C++, that successfully complies to these programming paradigms and that runs in different platforms, provided the availability of a compiler, e.g. the *cl* compiler in Windows (www.microsoft.com /express/download) and *gcc* in Linux (http://gcc.gnu.org). Both of these compilers are command line and, to ease their usage, the compilation and linkage processes are done with the *nmake* utility in Windows and with the *make* utility in Linux. In order to relieve the programmer of these platform-dependant issues, there exists CMake (www.cmake.org, [4]), an open-source multi-platform make system. CMake allows the user to prepare the project in a generic form for both Windows and Linux, relieving him/her from the task of configuring it for each platform. This helps the applications to be really multi-platform since the projects for compiling and linking them are easily obtained for each platform from the instructions coded using a script language in a text file called CmakeList.txt.

Control version system: Subversion SVN (https://sub-version.tigris.org, [5]) is a control version system that provides a good environment to store and restore all successive versions of the files in the project all along its development. It is useful for tracking and debugging purposes and essential when several programmers are involved. SVN provides a set of tools for common administrative server tasks and client operations. A choice on the server side is OpenSVN (https://opensvn.csie.org), an open-source initiative for open-source projects. For the client side, TortoiseSVN (http://tortoisesvn.net) is a Windows shell extension which integrates into Windows Explorer. For Linux environments, kdesvn (http://kdesvn.alwins-world.de) is a high integrated application for the KDesktop (similar tools can be found for other Linux desktops).

These tools can be complemented by others like those that allow the comparing of two versions of the same file to track and revise the changes (e.g. winmerge at www.winmerge.org or KDiff3 at http://kdiff3.sourceforge.net).

Documentation system: Doxygen is a multi-platform documentation system for several languages like C++ and Java. It is really easy to use since the programmer only needs to add some comments within the code with a special marks, and it automatically generates the documentation in both Html and LaTeX.

B. Graphic Rendering and GUIs

Any software application requires a GUI to interchange information with the user. A good multi-platform library for generating GUIs is Qt (http://trolltech.com/products/qt), since it is currently used in a great variety of open-source and commercial applications, and it has more than 10 years of developing experience. Moreover, Qt has a friendly GUI designer tool (QtDesigner) based on drag and drop that helps in this creative process.

For the graphic rendering of 3D models a good open-source multi-platform option is Coin3D (www.coin3d.org). This graphics library is based on OpenInventor sources delivered from Silicon Graphics and it is currently the implementation adopted for all Debian Linux distributions. Coin3D provides an interface similar to a CAD software. It can manage either basic models like spheres, cones and cubes, or general forms modeled with triangular meshes. Coin3D must work together with a GUI library, that can be either Qt, Motif/Xt, Windows or MacOs (for this interaction, SoQt or Quater, SoXt, SoWin and Sc21 libraries are, respectively, required).

Any 3D models can be described as a text file through Inventor or VRML languages (http://www.w3.org/ MarkUp/VRML). They describe a scene graph with high level objects like cubes, cylinders, cones or spheres or any grouping of these, as well as triangular meshes. Also, property nodes are introduced to affect the way shapes are drawn. Coin3D can load the models of a scene described with either Inventor or VRML files. A useful tool to rapidly build these scenes is Coindesigner (http://coindesigner.sourceforge.net), since it allows to write simple 3D scenarios just using drag and drop and visualizing the changes on the fly, i.e. choosing and easily configuring components from the Inventor/VRML toolkit and instantly adding them in the scene.

For the complete description of a robotic scene, however, the XML language can be very useful (http://sax.source-forge.net). The XML files can contain information of the robot and of the obstacles (VRML or Inventor files, names, scales, positions and orientations), and any other aspect related to the definition of a planning problem, like the initial and goal configurations. The Qt library has the capability to easily process XML files, and other interesting features like string manipulation or file reading.

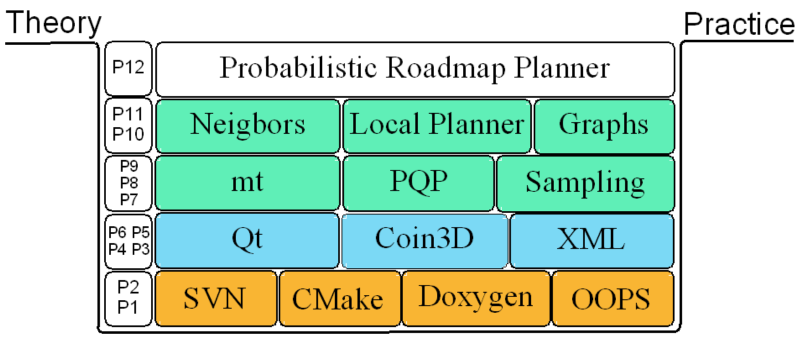


Fig. 1. Concepts covered by the set of assignments.

C. *Computational Geometry Tools*

Currently, sampling-based methods are the most commonly used methods in motion planning problems. These methods consist in the generation of collision-free samples of configuration space (Cspace) and in their interconnection with free paths, forming either roadmaps (PRM [6]) or trees (RRT [7]). Some of the needs to develop a sampling-based motion planner come from the computational geometry field: a) the location of objects; b) the generation of samples; c) the determination of the free or collision nature of the samples; d) the search of nearest neighbors, e) graph representations and search algorithms.

The specification of the position and orientation of objects in space can carry many problems if not managed correctly, e.g. due to inefficient representations that may lead to numerical errors. The mt library [8] helps in this direction by providing classes to properly define these representations.

Samples can be generated using either a random or a deterministic sequence. Random samples are generated by randomly setting the values of the configuration coordinates. These values are usually obtained using a pseudo-random number generator with long period and good statistical acceptance, like the one provided by [9] that gives better results than the *rand* function of the standard C library. Deterministic sequences provide samples giving a good incremental and uniform coverage of Cspace. The Halton sequence is one of the best low-discrepancy sequences (http://people.scs.fsu.edu/∼burkardt, [10]).

Collision detection procedures check if the objects of a scene are in collision or not (i.e. if the robot at a given configuration collides with the obstacles or not). Several open-source collision libraries are available on the internet. A good alternative for its simplicity is PQP, the Proximity Query Package (www.cs.unc.edu/geom/SSV, [11]). PQP is currently being used by many motion planners. It is based on quick and reliable algorithms to find collisions between objects described by triangular meshes. These meshes can be obtained from VRML or Inventor models using a conversion procedure provided by the Coin3D library. PQP can also answer separation distance queries.

In order to connect samples to capture the connectivity of the free configuration space, it is necessary to find, for each sample, which are its nearest neighbors. This can be efficiently solved using the ANN library (www.cs.umd.edu/∼mount/ANN, [12]). ANN is a library written in C++, which supports data structures and algorithms for both exact and approximate nearest neighbor searching in arbitrarily high dimensions. An adaptation of the ANN library to different topologies of Cspaces, with its corresponding metrics is available at msl.cs.uiuc.edu/∼yershova, [13].

To deal with graph data structures a good alternative for its efficiency is the Boost Graph Library (www.boost.org/libs/graph, [14]). This library is implemented using the generic programming paradigm. It provides a standardized generic interface with template search algorithms like Breadth First, Depth First, or Uniform Cost, and other algorithms that allows access to a graph's structure. They can be easily tailored to any particular application.

III. PROGRAMMING ASSIGNMENTS

A set of twelve programming assignments are proposed to be used as a practical counterpart of a robot motion planning course. It is assumed that: a) the course has a duration of one semester and that it introduces, among

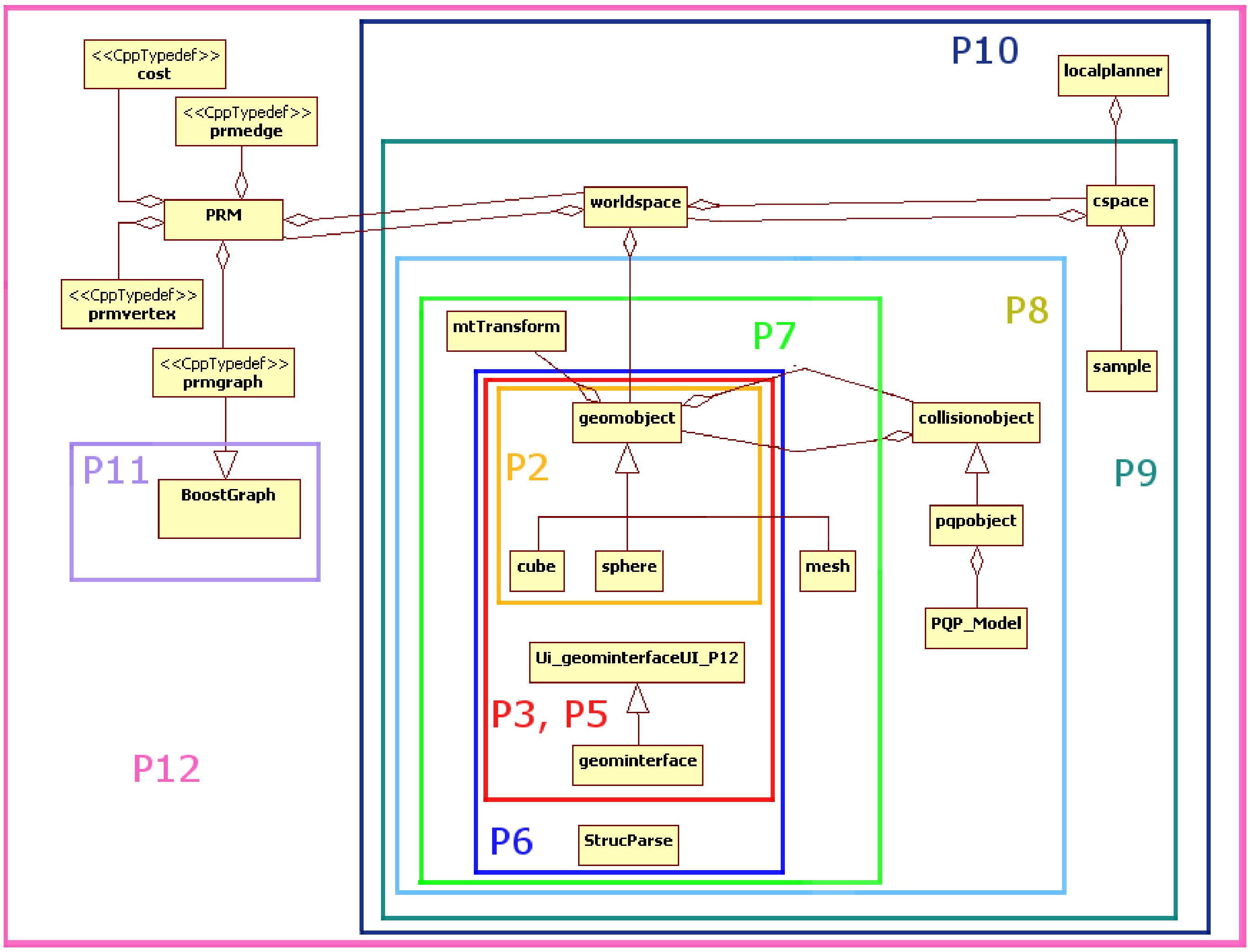


Fig. 2. UML graph of the whole project.

others, the concepts of configuration space, sampling-based techniques and probabilistic roadmaps; b) the students have good C++ programming skills.

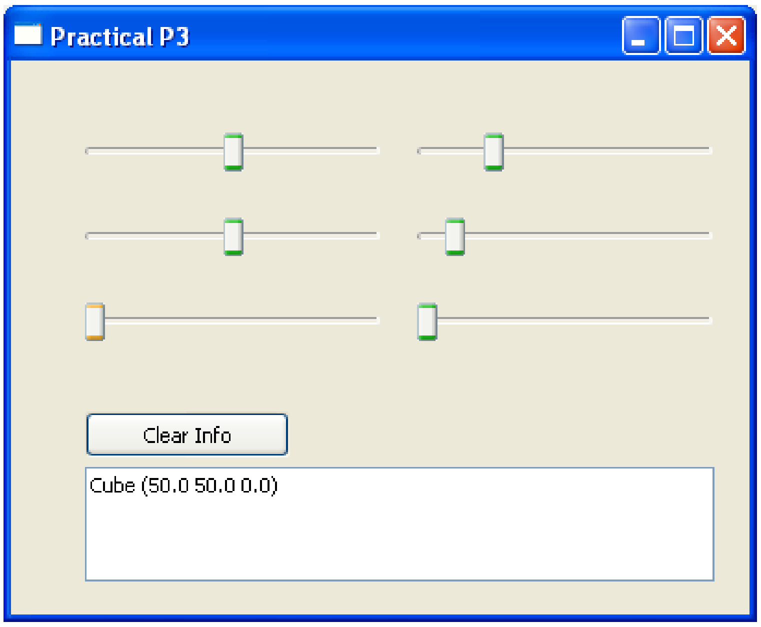
As shown in Fig. 1, the programming assignments incrementally cover all the basic aspects needed to build a probabilistic roadmap planner (PRM). Fig. 2 shows the UML graph of the PRM where the different classes that give its functionality are grouped depending on the assignment where they are introduced. The following subsections present each of these assignments showing for each one its aim, the material provided and the aspect it illustrates, and the assignment proposed to the students based on the implementation of some minor modifications that allow them easily getting a quick insight of it.

Prior to the start of the programming assignments, the students must install the following software into their computers: CMake, TortoiseSVN (if Windows OS is used), Doxygen, Coin3D, Qt4, SoQt, mt, PQP, Boost Graph Library and CoinDesigner2. The downloads and installation tips, as a well as the twelve programming assignments, are freely available at http://iocnet.upc.edu/usuaris/JanRosell/EducationalTools.html.

A. *Assignment P1: Learning how to use SVN and CMake*

Aim: The aim of this first programming assignment is to get the students used to the control version and the generation of the programming projects using CMake.

Material provided: Some simple demo examples.



Cube (50.0 50.0 0.0)

Fig. 3. GUI of assignment P3: A simple Qt interface with sliders that change the reference point of two objects of a scene.

Assignment: The students must perform a checkout of the whole set of assignments posted on the SVN server repository, and execute CMake to create the projects, either in Linux or Windows, of the examples posted as P1. Finally they must simply compile, link and execute the examples.

B. Assignment P2: Documenting with Doxygen

Aim: The aim of this assignment is to learn how to document using the Doxygen documentation system.

Material provided: A simple project that introduces an abstract class called geomobject and two derived classes: cube and sphere. This classes are organized in a separate folder and compiled as a library called libGeometry. It illustrates the concepts of OOP polymorphism and generic programming using templates.

Assignment: Extend the class hierarchy with classes cone and cylinder and document them using Doxygen.

C. Assignment P3: A simple Qt interface

Aim: The aim of this assignment is to use Qtdesigner to modify a GUI and learn how the events of the interface are managed with the slots mechanism.

Material provided: The P2 project enhanced with a Qt interface composed of six sliders and a text box. The sliders change the reference point of two objects instantiated from classes cube and sphere (Fig. 3).

Assignment: Complete the Qt interface by adding sliders to change the information of two new objects instantiated from classes cone and cylinder.

D. Assignment P4: Getting an Inventor/VRML insight

Aim: The aim of this assignment is to learn the structure of a graphic scene described with Inventor or VRML.

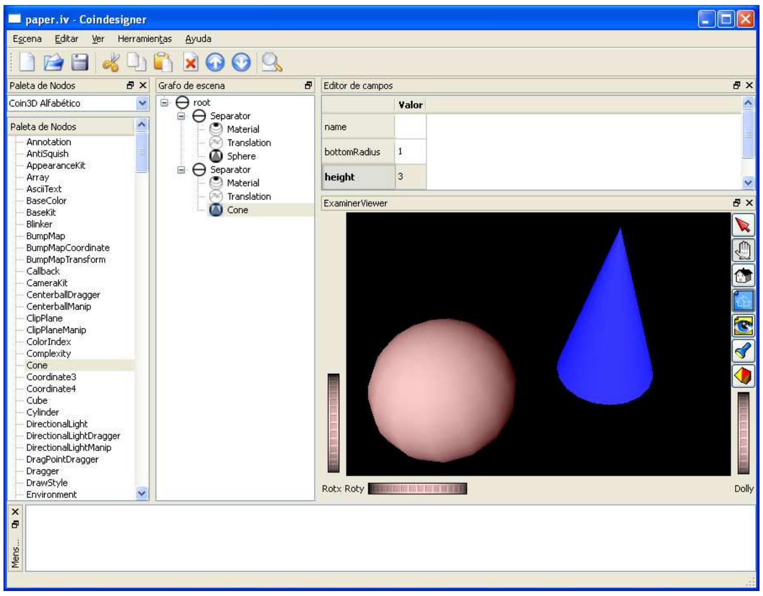
Material provided: A simple scene that includes shape nodes, property nodes and group nodes.

Assignment: Open the scene using Coindesiger2 (Fig. 4) and interactively edit the three types of nodes while monitoring the scene: a) change the geometry of a shape node; b) change the values of a transform and c) add a child to an existing node.

E. Assignment P5: Managing graphic scenes with Coin3D

Aim: The aim of this assignment is to learn how a graphic scene is rendered within a GUI based on Qt.

Material provided: The P3 project where the GUI has been enhanced with a 3D viewer showing the two objects of the scene (Fig. 5).



	Valor
name	
bottomRadius	1
height	3

Fig. 4. Inventor scene edited and rendered by Coindesigner2.

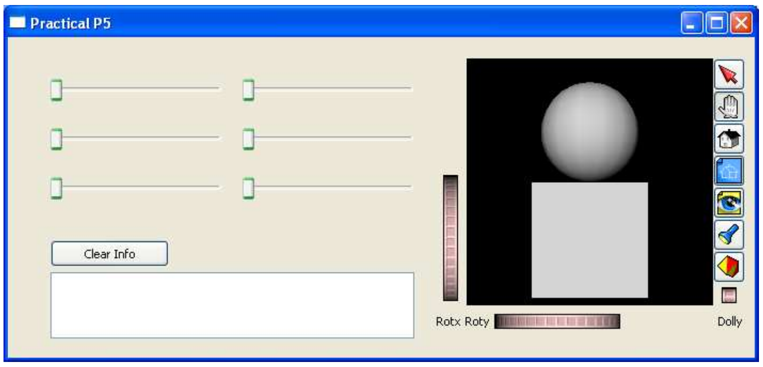


Fig. 5. GUI of assignment P5: The scene is graphically rendered using Coin3D.

Assignment: Complete the classes cone and cylinder introduced in assignment P3 in order to provide them with graphical capabilities.

F. Assignment P6: Input data with XML

Aim: The aim of this assignment is to learn how to describe a scene using XML.

Material provided: The P5 project enlarged with a class called structparse devoted to loading a scene described by an XML file. Fig. 6 shows part of one of the XML files provided describing the scene with cubes and spheres shown in Fig. 7.

Assignment: Modify the class structparse to correctly manage objects from the new classes cylinder and cone.

G. Assignment P7: Moving a robot: translating and rotating an object using the mt *library*

Aim: The aim of this assignment is to get used to the mt library to define the position and orientation of 3D objets, as well as the interpolation motions between configurations.

Material provided: The P6 project enhanced as follows. Class geomobject now has as a parameter a transformation defined by class Transform of the mt library. The GUI allows to modify the position and orientation of the movable object, and to interpolate the motion from the current configuration to the initial one using the

```
<?xml version="1.0" encoding="UTF-8"?>
<Scene>
      <Geomobject>
            <File objectfile="objects/greencube.iv">
                   Object File
            </File>
            <Type T="cube">
                   Type of geomobject
            </Type>
            <Configuration TH="0.0" WZ="1.0" WY="0.0" WX="0.0" Z="0.0" Y="0.0"
      X="0.0">
                   Object Position and Orientation
            </Configuration>
            <Properties p1="1.0" p2=" " p3=" ">
                   p1 contains Cube side - p2 and p3 not used
            </Properties>
            <Scale sf="1.0">
                   Scale factor to be applied to the ivfile
            </Scale>
      </Geomobject>

      . . .

      <Geomobject>
            <Type T="sphere">
                   Type of geomobject
            </Type>
            <Configuration TH="0.0" WZ="1.0" WY="0.0" WX="0.0" Z="0.0" Y="1.0"
      X="0.0">
                   Initial Configuration of the robot
            </Configuration>
            <Properties p1="0.5" p2=" " p3=" ">
                   p1 contains Sphere radius - p2 and p3 not used
            </Properties>
      </Geomobject>
</Scene>
```

Fig. 6. XML file defining the objects of a scene.

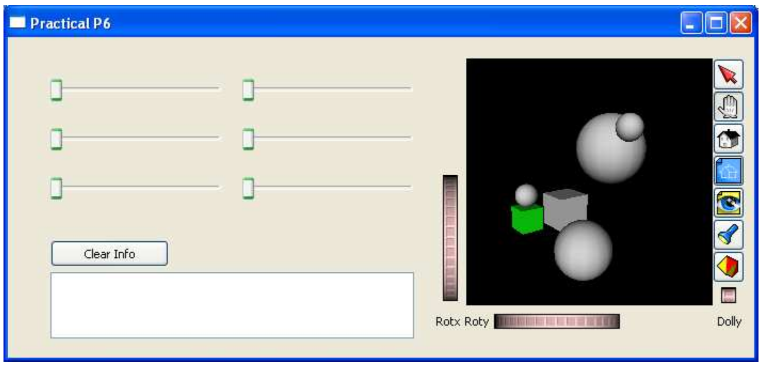


Fig. 7. GUI of assignment P6: A scene read form an XML file is graphically rendered.

Reset Scene button. Also, a new class mesh that inherits from class geomobject is added to handle any shape described by an Inventor file (Fig. 8).

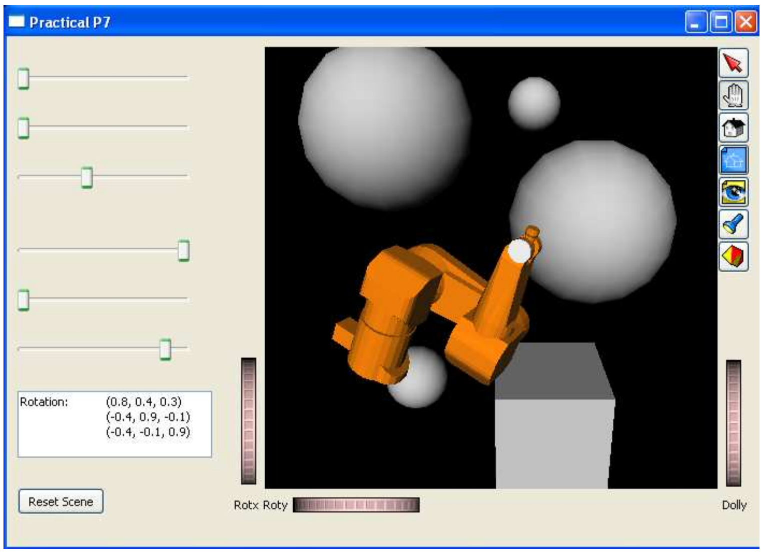
Assignment: Add two push buttons, one to store the current configuration of the movable object and the other to interpolate the motion from the current configuration towards the last stored configuration.

H. Assignment P8: Performing collisions with PQP

Aim: The aim of this assignment is to learn how collision queries are done using the PQP library and how the PQP models can be obtained from the Inventor models.

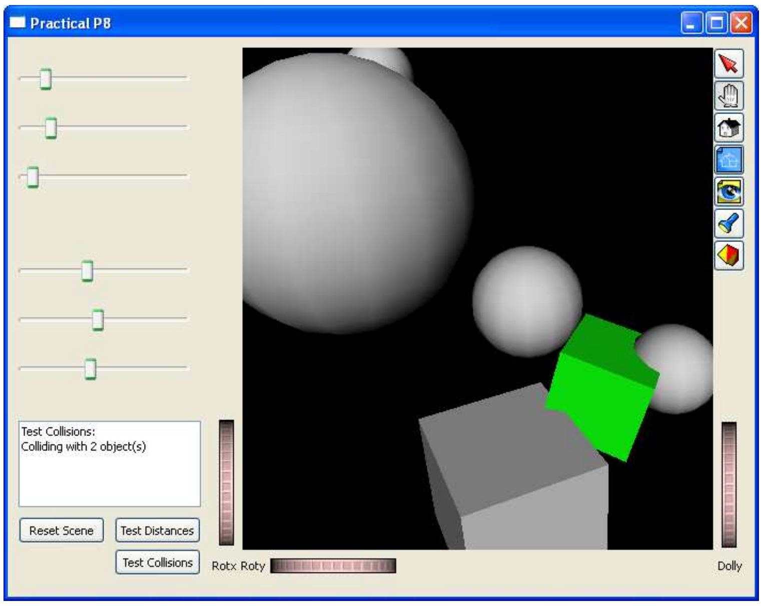
Material provided: The P7 project enlarged with class collisionobject and its derived class pqpobject (other collision libraries should derive its own classes). This classes are organized in a separate folder and compiled as a library called libCollision. Each instance of class geomobject has now a pointer to an instance of class collisionobject that handles the collision queries with all other instances of class collisionobject (Fig. 9).

Assignment: Modify the GUI to allow the selective collision queries.



Rotation: (0.8, 0.4, 0.3)
(-0.4, 0.9, -0.1)
(-0.4, -0.1, 0.9)

Fig. 8. GUI of assignment P7: Sliders allow to change the position and orientation of a free-flying robot defined by a triangular mesh.



Test Collisions:
Colliding with 2 object(s)

Fig. 9. GUI of assignment P8: Collisions and distance queries are answered using the PQP library.

I. Assignment P9: Sampling Cspace

Aim: The aim of this assignment is to focus the attention on the modeling of the workspace, of the configuration space, and on the sampling procedures.

Material provided: The P8 project enlarged with classes worldspace, cspace and sample. This classes are organized in a separate folder and compiled as a library called libCspace. The GUI is organized in two tabs. The first tab with the previous utilities and two new push buttons to start the sampling; the second tab allows the scanning of all the samples showing the movable object at the sampled configuration and the information about its nature: collision or free (Fig. 10).

Assignment: Implement the deterministic sampling using the Halton sequence [10].

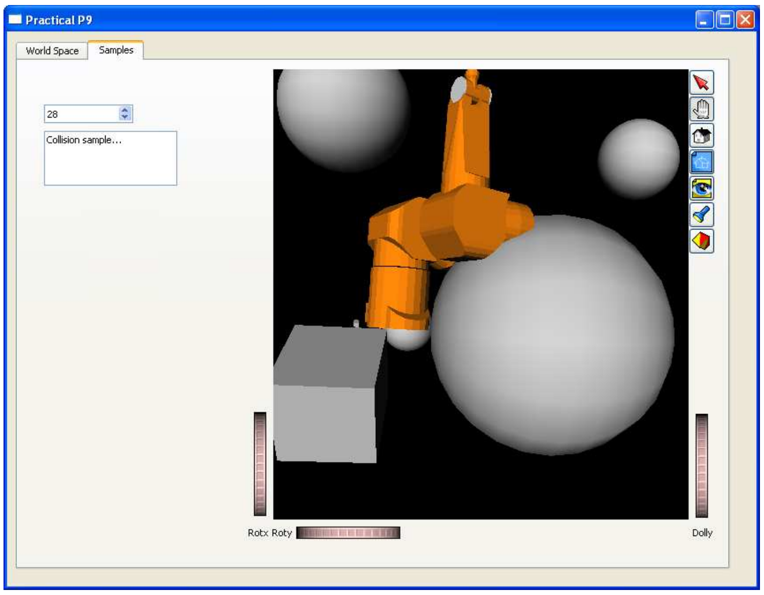


Fig. 10. GUI of assignment P9: The current tab shows the free-flying robot at one of the sampled configurations and informs that it is colliding.

J. Assignment P10: Finding neighbors and implementing a local planner

Aim: The aim of this assignment is to focus on two of the basic needs for a roadmap-based motion planner, namely the search of nearest neighbors and the implementation of a local planner.

Material provided: The P9 project enlarged with the algorithm to search the nearest neighbors (done with the brute-force method) and the local planner implemented with a new class called localplanner, that uses the incremental strategy [15]. The GUI allows to set an initial and a final sample and test if the edge linking them is free or not; it also answers queries for the samples' neighbors (Figure 11).

Assignment: Implement the local planner using the binary method [15]; as an advanced assignment implement the search of the nearest neighbors using the ANN algorithm [13].

K. Assignment P11: Managing graphs with the Boost Graph Library

Aim: The aim of this assignment is to get a minimum insight of the Boost Graph Library.

Material provided: A simplified version of the A* example provided by the Boost Graph Library.

Assignment: Modify the example loading a different graph and changing the heuristic used.

L. Assignment P12: My first PRM

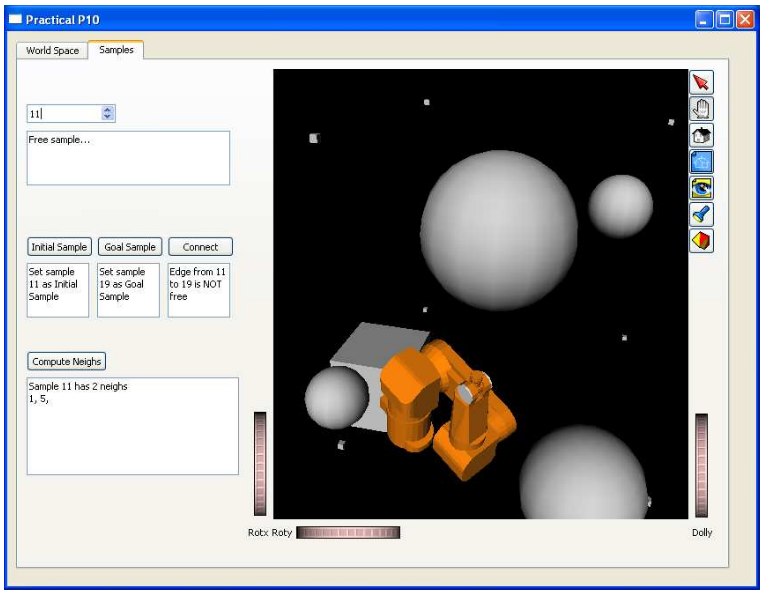
Aim: The aim of this assignment is to have a global view of all the parts involved in a probabilistic roadmap planner and their interactions.

Material provided: The P10 project enlarged with the class PRM that implements the planer using all the functionalities of the graphs provided by the Boost Graph Library. Figure 12 shows the third tab added to the GUI where the user can select the start and goal nodes and, if a path exists, visualize the object moving along it.

Assignment: Modify the PRM introducing the options implemented in assignments P9 and P10, namely the deterministic sampling and the binary method for the local planner.

IV. CONCLUSIONS

The intuition needed to make contributions in any research field must be strongly based on a good knowledge of all its related subjects. In the scope of robot motion planning this also includes all the aspects of robot simulation. This paper has been written to provide an aid for students of robot motion planning courses to get a quick mastering of the basic issues related with robot simulation. The paper first proposed a set of software



```
11
Free sample...

Set sample 11 as Initial Sample
Set sample 19 as Goal Sample
Edge from 11 to 19 is NOT free

Sample 11 has 2 neighs
1, 5,
```

Fig. 11. GUI of assignment P10: It allows to test if an edge connecting two samples is free or not and to compute the neighbors of the current sample.

tools based on multi-platform open source code philosophy. For a smooth learning curve towards the mastering of these tools, the paper then presented a set of twelve software assignments that incrementally present, in a simple way, all the basic concepts needed. The proposal is currently being used with success by the authors to teach "Planning in Robotics" within the PhD Program "Automatic Control, Robotics and Computer Vision " at the Technical University of Catalonia.

REFERENCES

[1] A. Pérez and J. Rosell, "A roadmap to robot motion planning software development," *Computer Application in Engineering Education*, vol. Accepted, June 2008.

[2] G. Booch, *Object-Oriented Analysis and Design with Applications*, 3rd ed. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 2004.

[3] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language*. Addison Wesley, 1999.

[4] K. Martin and B. Hoffman., *Mastering CMake: A Cross-Platform Build System*. Kitware Inc., 2005.

[5] B. Collins-Sussman, B. Fitzpatrick, and M. Pilato, *Control version with Subversion*. O'Reilly, 2007.

[6] L. Kavraki and J.-C. Latombe, "Randomized preprocessing of configuration for fast path planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 3, May 1994, pp. 2138 – 2145.

[7] J. Kuffner, J.J. and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, 2000, pp. 995–1001.

[8] A. Rodríguez, "The mt library," Technical report IOC-UPC-25-2008, Tech. Rep., June 2008.

[9] S. Rabin, *AI Game Programming Wisdom 2*. Charles River, 2003.

[10] J. H. Halton, "On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals," *Numerische Mathematik*, vol. 2, no. 1, pp. 84 – 90, December 1960.

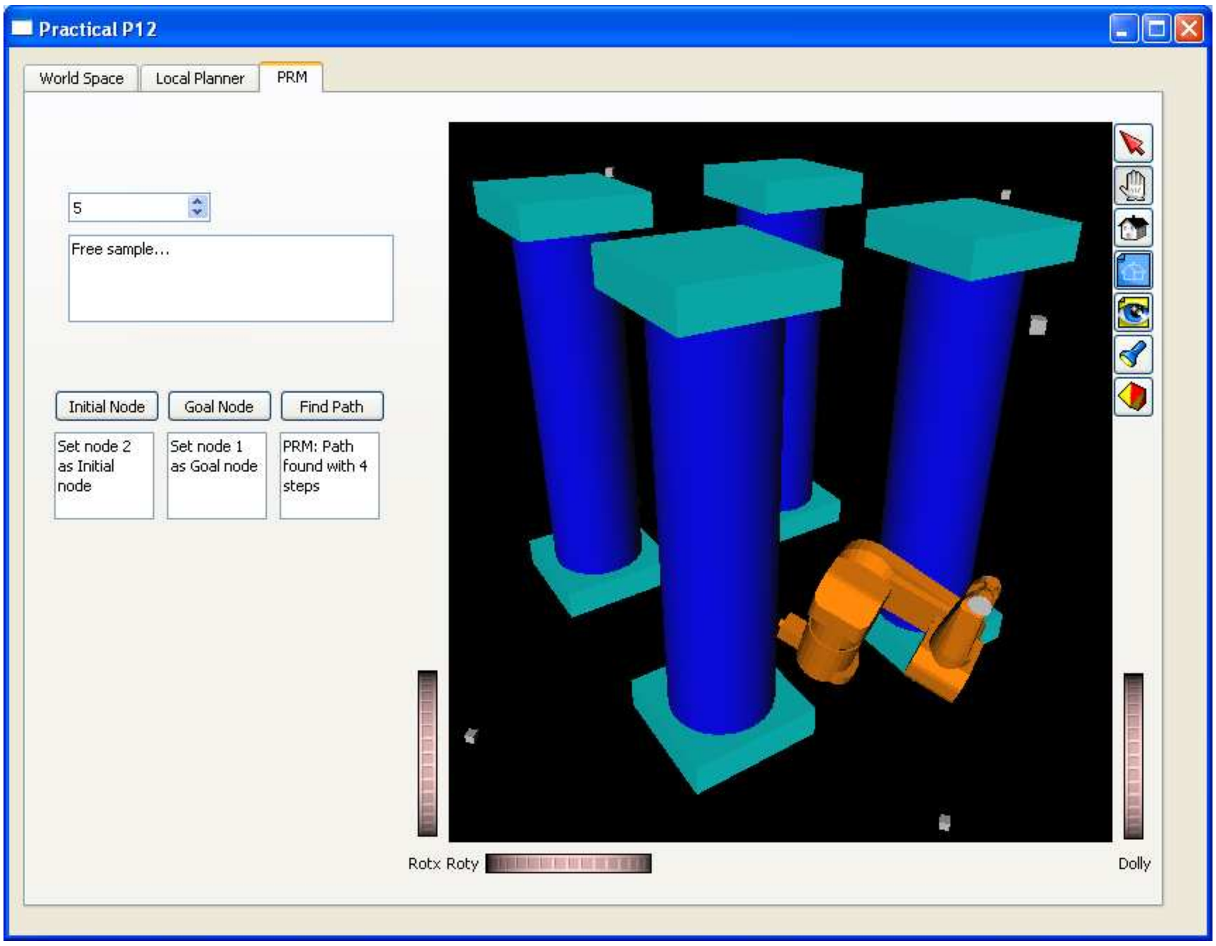
[11] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha, "Fast proximity queries with swept sphere volumes," in *IEEE International Conference on Robotics and Automation (ICRA)*, April 2000, pp. 3719–3726.

[12] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, "An optimal algorithm for approximate nearest neighbor searching fixed dimensions," *J. ACM*, vol. 45, no. 6, pp. 891–923, 1998.

[13] A. Yershova and S. M. LaValle, "Improving motion-planning algorithms by efficient nearest-neighbor searching," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 151–157, February 2007.

[14] L.-Q. Lee, J. G. Siek, and A. Lumsdaine, "The generic graph component library," in *14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications.*, New York, 1999, pp. 399–414.

[15] M. H. O. Roland Geraerts, "Sampling and node adding in probabilistic roadmap planners," *Robotics and Autonomous Systems*, vol. 54, p. 165173, 2006.



5

Free sample...

Set node 2 as Initial node

Set node 1 as Goal node

PRM: Path found with 4 steps

Fig. 12. GUI of assignment P12: The path between two nodes of the PRM can be computed and visualized.