CSCI–4190 Introduction to Robotic Algorithms, Spring 2002
Assignment 1 details, Tuesday January 22

This document contains the following sections:

- Corrections and clarifications from the assignment handout
- Using the CS department computers
- Getting started
- A guide to the support code
- Written questions

Corrections and clarifications from the assignment handout

- Instead of the motion planning algorithm given in the assignment handout (in the "Creating a quadtree representation..." section), use something more sophisticated, such as the algorithm given in Section 3.2.1 of the course notes. The algorithm in the assignment handout will stop at the first E-channel it finds; I want you to implement something better than that. It does not necessarily have to be the algorithm in the course notes, but in general it should produce "equally intelligent" paths.
- Here's an idea of how I'd like to interact with your program:
  - When the program starts, it should display the workspace and the start and goal locations.
  - Type "c" to compute and display the configuration space.
  - Type "p" to compute and display the initial quadtree.
  - Repeatedly type "p" to compute and display successive refinements of the quadtree until your motion planning algorithm decides it is done.
  - Type "p" one more time to compute and display the adjacency graph for the final quadtree decomposition.
  - Type "p" one more time to display the E-channel (or alternatively, the path through the adjacency graph, preferrably overlaid on the quadtree decomposition.)
  - Type "p" one more time to compute and display the path through the E-channel. (If there is no path, it can just do nothing.)
  - Type "q" to quit.

  Your program need not follow the exact steps above, but I do want to see at least the workspace, the configuration space, successive refinements of the quadtree, the final adjacency graph, and the planned path for the robot. You can also consider displaying an E-channel, M-channel, and/or adjacency graph after every iteration or just at the end.

  Your program should print (to `cout`) instructions on how to interact with your program.
- A good choice for the minimum allowable cell size is to use some fraction of the robot size, e.g. 1/20 of the longer dimension of its bounding box.
- I'll provide instuctions for turning in your program at a later time.
- Comment your code. In most cases, I expect to at least briefly look through your code. If I need some clarification of what you are actually doing beyond what is visually displayed or described in the written questions, I will look through your code. Make it easy for me to grade this assignment!

Using the CS department computers

For this assignment, you will have to use the CS department SunOS (UNIX) computing environment.

CS accounts

All of you (as of last Thursday) have CS accounts; most of you already had a CS account.

If you ever need to have your password reset, contact `labstaff@cs.rpi.edu`. You can also stop by Lally 301 during business hours. Be sure to bring ID with you. (You might want to call x2842 to make sure they are there first.)

Information on using CS machines

You can find basic information on using CS machines at:

`http://www.cs.rpi.edu/guide/machines/`

Physical access to machines

There are two labs with machines that you can use:

- Amos Eaton 117 (the OOT lab) — contains 16 Sun Ultra 10 workstations. This lab is open Monday to Friday from 9am to 5pm; however, if there is someone already in the lab at other times, you can probably get them to let you in.
- Amos Eaton 215 — contains ?? SunRay workstations/terminals. This lab is open (in theory) 24 hours a day, but you cannot use this room while scheduled classes are there. The schedule of classes is posted on the door, but I'm not sure it's accurate. I'll keep you posted.

Please note that the Amos Eaton building is locked over the weekend and at night on weekdays (generally at sometime between 7pm and 10pm). If you try all the doors on all sides of the building or wait patiently by the front door for someone to leave, you can probably get in.

The CS department also has a lab in Amos Eaton 217, but this is reserved for CS graduate students only.

Remote access to machines

You can access the CS machines remotely by connecting to:

`solaris.remote.cs.rpi.edu`

This will actually connect you to one of 16 computers. Please do not connect to specific machines by name. Note that you will need to use a solaris machine in order to do this assignment.

These computers do not accept unencrypted connections; you must use some version SSH to login to these machines. If you wish to connect under Windows, RPI laptops should already have SecureCRT, or you can get a free version of TeraTerm with SSH extensions. See `http://www.cs.rpi.edu/lab/software` for links to this software. The connections from Windows machines probably won't be able to display graphics, but you can at least compile and edit code.

Working on other platforms

The packages used in the support code (GLUT, OpenGL, and CGAL) are available on other platforms, including Windows, but currently I do not have time to support other platforms. There is no reason in principle why you can't get the support code to compile on other platforms; this should be pretty straightforward under Linux, probably not too difficult under FreeBSD, and probably a bit challenging under Windows. The support code will grow in future assignments to include at least CORBA (which is also available on other platforms).

Even if you work on another platform, your code must still compile and run on the CS machines.

Getting started

Login to a CS machine, get an "xterm" (left-click on the background and select from the menu), and type the following commands at the command prompt (`$`):

`$ mkdir ira`	create a directory for this class
`$ chmod 700 ira`	set permissions so only you can access the files
`$ cp -r /projects/ira/assign1 ira`	make a copy the support code
`$ cd ira/assign1`	
`$ ls`	list all the files
`$ gmake`	compile everything
`$ ./assign1 Problem1.txt`	run the program

When you run the program, it will read in the problem description and then pop up a window where it will draw the world. Type "q" to quit the program. (Make sure the mouse is in the window.)

To start looking at the code, type:

```
$ emacs&
```

This will start the GNU Emacs editor which will pop up in a window. (You could have also given it some filenames on the command line, before the ampersand.) GNU Emacs is the editor of choice on UNIX systems, and I suggest you invest the time to learn the basics. You can get a tutorial by typing `C-h t` (that's "control-h" followed by "t") in the Emacs window, or you can select "tutorial" from the pulldown "help" menu.

A guide to the support code

The support code provided includes functions and classes to read the problem from files, to represent the world, open a window and draw graphics, a geometric intersection routine, and a sketch of some classes which may be useful in writing your program.

All the source code is being released, and you can use it to write your program for this assignment however you like. However, I suggest you work within the structure of the support code I've provided. In particular, you should try not to change any of the files except for `motplanning.{h,cc}` and `assign1.cc`. If I have to release revised code to correct any errors, then you will be able to simply replace the other files. If there is some change that you'd want to make to these other files, you might want to consult me first, and I may incorporate in into a new release of the support code.

Graphics

The graphics support code is defined in the files `graphics.{h,cc}` and `drawable.h`. These files define a module whose public interface is specified in `graphics.h`. The main program in `assign1.cc` illustrates how to use this module.

The basic idea is that there is a global variable `objectLists` which is a list of lists of `drawableObject`s (actually, it is a pointer to a list of lists of pointers to `drawableObjects`). The `drawableObject` class, defined in `drawable.h` is a pure abstract class which defines an interface for all objects that are to be drawn on the screen. The `world.h` and `motplanning.h` files define classes derived from this class; they should be all that you need for this assignment.

The intention behind this design was that you would have lists of objects that need to be drawn, such as all the configuration space obstacles or a quadtree. You can then add (a pointer to) that list to `objectsLists`. This makes it easy when, for example, you want to remove the current quadtree that is being drawn and replace it with a new quadtree.

Your main program should set up the graphics, and then it must call the `runEventLoop()` method. All further action is triggered by an event received in response to the user's input. Currently, this is limited to keyboard events (typed characters). There is a provision to install your own keyboard handler in place of the default.

World representation

The `world.{h,cc}` files contain the `WorldBoundary`, `Obstacle`, `World`, `Robot`, `Path`, and `Location` classes; they should be reasonably self explanatory. The `mpproblem.{h,cc}` files contain the `MPProblem` class which will read the problem description (including the world and obstacles) from files.

A problem consists of a problem file, a world file, and a robot file. There is one example of each of these: `Problem1.txt`, `World1.txt`, and `Robot1.txt`. See the comments in these files for information about their format.

Writing your program

The main program is contained in the file `assign1.cc`. In the files `motplanning.{h,cc}`, I have partially sketched out `CObstacle` and `Quadtree` classes which you will probably find useful. I'd suggest that you put your code in these two files. You may want to put some code in additional files; you should be able to figure out how to include any additional files in the compilation and linking process by looking at the +Makefile+.

CGAL and geometric computations

The CGAL library (`www.cgal.org`) is used for geometric representation and operations. I will provide additional information on relevant CGAL functions on the course web page. In the files `utilities.{h,cc}`, there is a function which will return information about the intersection of a bounding box and a polygon. All the include files for CGAL and some important `typedef`s are found in the file `cgaldefs.h`.

Note on debugging

CGAL is a complex library that makes extensive use of templates. As such, it can produce voluminous compilation errors even for small mistakes. For example, I recently got two pages of errors just because of a forgotten semicolon!

I have the following suggestions for your debugging:

- Find the first error and see which file it occurs in (and at what line number). Go to the source code and see if there is an obvious problem. It's worth a little while just staring at the source code and thinking about it.
- Then look to see what the error is — if this involves a conversion that the compiler cannot make or if it involves a function for which the compiler cannot find a matching declaration, look to see what the types of the arguments are.
- You should probably just try to fix a handful of problems and then recompile instead of trying to go through all the errors that the compiler generates.

Written questions

1. Describe the motion planning algorithm you used, in particular how it differs from the basic algorithm given in the course notes for approximate cell decompositions. Also describe any relevant details, such as how you created the configuration space representation of the boundary, the cost function used in your search determines distance between two different size cells, and how you generated a path within an E-channel.
2. Is it acceptable for a FULL cell to be mistakenly labeled MIXED? Is it acceptable for a MIXED to be mistakenly labeled FULL? What are the implications on the correctness and completeness of the motion planning algorithm?
3. Why do we tend to use the A* search algorithm for searching graphs in motion planning algorithms instead of, for example, Djikstra's algorithm which finds the shortest path in any graph?
4. Suppose you had to modify your program to plan motions for a robot that can rotate. What would you have to change in your program? Describe and sketch solutions to the main problems you would encounter.