

## Homework 4 Appendix: Reading in Map Files

---

As noted in the Homework #4 assignment, you need to read in the map file into your program's memory. Your path planner will use this map for finding the robot's path. Note that the bitmaps we have been using are stored in PNG format, which makes use of data compression techniques. While this is great for saving file space, it isn't the easiest format to use for path planning. So, we're going to convert the bitmap files to the PNM ("Portable Any Map") format, which is often used for exchanging between different graphics file formats. [Since the purpose of this class isn't to learn about graphics file formats, we're going to skip over the details. If you want to learn more about these file formats, there is lots of documentation on the web.]

Fortunately, we have handy utilities that can do this conversion for us. The utility we need is called, amazingly enough, "pngtopnm". This utility is so simple to use, that I'm going to let you convert the hospital\_section.png file to the pnm version we'll call hospital\_section.pnm. Simply issue the following linux command:

```
linux> pngtopnm hospital_section.png > hospital_section.pnm
```

The resulting PNM file has 3 header lines, followed by the pixels defining the map. You can safely ignore the first and third header lines in the PNM file. The 2<sup>nd</sup> header line has handy information on the width and height of the image, in pixels. The rest of the file has a character for each pixel, representing white pixels with -1 and black pixels with 0. We'll convert this to 0's for free space and 1's for obstacles in our grid map. So, to read in this map file, I've written a little utility for you. Creatively, it is called inputMap(). You can find the electronic source on the class web site, here:

<http://www.cs.utk.edu/~parker/Courses/CS594-spring07/Assignments/inputMap.cc> I've also attached a hard copy of the function at the end of this document.

Let me point out one more detail. Note that the map is scaled using the "SCALE\_MAP" parameter. In this function, the effect of this parameter is to scale down the size of the map by a factor of 1/SCALE\_MAP. This effectively sets the grid size of the map that your planner will use. To determine the size of the grid, you need to compare the metric size of the map (as defined in your world file) with the pixel size of the map (as defined in the PNM file). In this case, the metric size of the world map is 40 meters wide by 18 meters high. The size of the PNM file, in pixels, is 1086 wide by 443 high. We can therefore calculate the number of pixels per meter as 1086pixels/40m wide by 443pixels/18m high, giving us 27.15pixels/m and 24.61pixels/m. Inverting this, we get 3.68cm/pixel and 4.06 cm/pixel. We'll approximate this as 4 cm/pixel.

If we make one grid cell per pixel, then our grid resolution is 4x4 cm. This is probably a finer-grained resolution than we need. We can double the grid size to 8x8cm (thus halving the resolution). This seems like a reasonable resolution for our purposes. Thus, we scale the map by a factor of 2 (by setting SCALE\_MAP to 2) in the inputMap() function.

The function is also set up to allow you to print out your scaled-down map to a separate file (called "scaled\_hospital\_section.pnm"). You can then use an image display program, such as *gimp*, to view the resulting file. You may find it handy to print out your map after you've grown obstacles, or even to display your planned path (perhaps by setting your path segments to '1', so that they display as a line in the whitespace), by printing out your result in the PNM format. Then you can have a look at what your program has generated using the *gimp* utility. This should be helpful for debugging purposes.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

#define SCALE_MAP 2
#define GRID_ROWS 1000
#define GRID_COLS 1000

// You can change this to use dynamic memory, if you like.
float gridMap[GRID_ROWS][GRID_COLS];

/*****
 *
 * Function inputMap converts the input map information into an
 * initial occupancy grid, which is stored in gridMap.
 *
 * (Here, the "output" parameter says whether to print out the scaled
 * map; output = 1 ==> yes, print; output = 0 ==> no, don't print.)
 *****/

void inputMap(int output)
{
    int i, j, m, n;
    char inputLine1[80], nextChar;
    int width, height, maxVal;

    ifstream inFile("hospital_section.pnm");

    /* Initialize map to 0's, meaning all free space */
    for (m=0; m<GRID_ROWS; m++)
        for (n=0; n<GRID_COLS; n++)
            gridMap[m][n] = 0.0;

    /* Read past first line */
    inFile.getline(inputLine1,80);

    /* Read in width, height, maxVal */
    inFile >> width >> height >> maxVal;
    cout << "Width = " << width << ", Height = " << height << endl;

    /* Read in map */
    for (i=0; i<height; i++)
        for (j=0; j<width; j++) {
            inFile >> nextChar;
            if (!nextChar)
                gridMap[i/SCALE_MAP][j/SCALE_MAP] = 1.0;
        }
    cout << "Map input complete.\n";

    if (output) {
        ofstream outFile("scaled_hospital_section.pnm");
        outFile << inputLine1 << endl;
        outFile << width/SCALE_MAP << " " << height/SCALE_MAP << endl
            << maxVal << endl;
    }
}
```

```
        for (i=0; i<height/SCALE_MAP; i++)
        for (j=0; j<width/SCALE_MAP; j++) {
            if (gridMap[i][j] == 1.0)
                outFile << (char) 0;
            else
                outFile << (char) -1;
        }
        cout << "Scaled map output to file.\n";
    }
}

int main(int argc, char *argv[])
{
    inputMap(1); // Here, '1' means to print out the scaled map to a file;
                // If you don't want the printout, pass a parameter of '0'.
}
```