Written Part | Programming Part | Extra Credit | Submission Details | Support Code: General Description | Support Code: Install/Compile/Run

Assignment 1: Bug Algorithms (with support code)

Due Date: Feb 17th, 2010 @ 11:59PM EST

Written Part (30 pts)

- Following the proofs for Bug1, write proofs for the following lemmas:
  - **(5pts)** Bug2 meets only a finite number of obstacles. Moreover, the only obstacles that can be met are those that intersect the straight-line segment from init to goal
  - **(8pts)** Bug2 will pass any point of the i-th obstacle boundary at most $n_i/2$ times, where n_i is the number of intersections between the straight line (init, goal) and the i-th obstacle
- Design your own bug algorithm, which we will call MyBug. MyBug should be different from the algorithms covered in class.
  - **(8pts)** Explain the general idea and the basic steps of your algorithm
  - **(2pts)** Draw a simple scene and trace your algorithm
  - **(4pts)** Draw a scene where your algorithm does better than Bug0 (if you cannot, then you need to redesign your algorithm)
  - **(3pts)** What are the strengths and weaknesses of your algorithm in comparison to the other Bug algorithms?

Programming Part (70pts)

The robot is modeled as a disk with a certain radius. The robot is equipped with a simple range sensor. When obstacles are within the sensing range, the sensor can determine the minimum distance to the obstacles and the nearest point to the obstacles from the center of the robot. The sensing range is always greater than the robot radius. The robot should never collide with the obstacles. You need to implement the following functionality:

- **(5pts)** MoveTowardGoal
- **(15pts)** FollowObstacleBoundary
- **(35pts)** [CS336: Bug1 **or** Bug2] [CS436: Bug1 **and** Bug2]
- **(15pts)** Your own bug (MyBug) algorithm

Support code is provided, which should greatly facilitate implementation. Read more in the **SupportCode** section.

Extra Credit (applicable to both CS336 and CS436)

- Student with the best MyBug algorithm gets **+10pts**
- Student with the 2nd best MyBug algorithm gets **+5pts**
- Student with the 3rd best MyBug algorithm gets **+3pts**
- **Note:** Bug algorithms will be ranked based on the overall path length produced for three new test scenes, which will be similar to the seven scenes that are provided to you as part of the support code.

When to Submit?/What to Submit?/How to Submit?

- **Due Date: Feb 17th, 23:59PM EST**
- Submit answers to written part. PDF preferred, doc accepted.
- Submit your code.
- Put everything in one directory named {CS336 or CS436}_Ass1_{LastName}{FirstInitial}, e.g., CS336_Ass1_PlakuE
- Compress the directory (tar.gz or zip) and email it to **cs336/AT/cs.jhu.edu**

Support Code -- General Description

Support code is provided for this assignment to facilitate your implementation (see documentation for detailed descriptions). The main classes in the support code are the following:

- **Simulator** : simulates robot motion and sensor [already implemented]
- **Graphics** : provides graphical display and allows for problem setup [already implemented]
- **BugAlgorithms** : interface for various bug algorithms [you need to implement it]

The program flow is as follows:

- **Graphics** uses a timer to call **BugAlgorithms**
- **BugAlgorithms** is responsible for selecting the appropriate displacement of the robot center along the x and y axis, i.e., determining how the robot should move. **BugAlgorithms** has access to **Simulator**.
- **Simulator** provides access to the current robot position, goal position, and sensor readings. You should use this information to implement the various parts in your programming assignment.
    - m_simulator->GetRobotRadius(): return robot radius
    - m_simulator->GetRobotCenterX(): return x position of robot center
    - m_simulator->GetRobotCenterY(): return y position of robot center
    - m_simulator->GetSensorRadius(): return sensor range
    - m_simulator->TakeSensorReading(xmin, ymin): This function returns the minimum distance from the current robot position to the obstacles. Upon return, xmin and ymin contain the x-coord and y-coord of the obstacle point that achieves this miminum. If the obstacles are outside the sensing range, then the sensing range is returned and xmin and ymin are set to HUGE_VAL
    - m_simulator->GetGoalRadius(): return goal radius
    - m_simulator->GetGoalCenterX(): return x position of goal center
    - m_simulator->GetGoalCenterY(): return y position of goal center
    - m_simulator->HasRobotReachedGoal(): return true iff robot has reached goal
    - m_simulator->GetDistanceFromRobotToGoal(): return distance from robot center to goal center

  Simulator is considered as an input to your implementation of the bug algorithms. As such, you should not make changes to **Simulator**. In particular, your bug algorithms should only access the public functions of **Simulator**. If, however, you see that the functionality provided by the **Simulator** class is not sufficient to implement your bug algorithms, please contact the support staff for this project (see main webpage for contact information). In your email, succinctly describe the functionality you intend to add to this class and the reasons as to why it is necessary for your implementation of the bug algorithms.

- **Graphics** then uses **Simulator** to apply the selected displacement and move the robot accordingly.

Support Code: Installing/Compiling/Running

- Install a compiler

  - Unix/Linux: Install g++
  - MacOS: Install gcc. The easiest way is to download the Apple Developer Tools from http://connect.apple.com (free, but requires registration). You can also install gcc through a package manager such as MacPorts or Fink
  - Windows: Install a compiler such as Microsoft Visual C++ Express Edition 2008 (it's free)

- Install GLUT
  - Unix/Linux: Use your package manager if not already installed
  - MacOS: All set, should already be installed
  - Windows: A pre-compiled version of GLUT can be found here.
    - Download glut-3.7.6-bin.zip
    - Extract the files from the archive
    - Put glut32.dll inside C:/Windows/system32
    - Put glut.h inside C:/Program Files/Microsoft SDKs/Windows/"latest version"/Include/GL
    - Put glut32.lib inside C:/Program Files/Microsoft SDKs/Windows/"latest version"/Lib

- Install CMake. CMake is a Makefile generator and is used to create makefiles for different compilers and operating systems.
  This step is not necessary if using the lab machines, since CMake is already installed there.

- Download support code Assign1.zip and unzip

- Compile

```
#Linux/Mac OS X
         cd Assign1
         cmake .
         make

#Windows (Microsoft Visual Studio)
        open command prompt
        cd Assign1
        cmake -G "Visual Studio 9 2008"
        go to the Assign1 folder and double-click on the .sln file
        build the project
```

  If OpenGL, GLU, and GLUT libraries were not found, then something is wrong with your installation of these libraries. If you are working in the lab machines, the compilation should not give you any problems.

- Run the code (from the prompt command inside directory Assign1)

```
#Linux/Mac OS X
       ./bin/RunBug obstacles1.txt

#Windows
       open command prompt
       cd Assign1
       bin\Debug\RunBug obstacles1.txt
```

  where obstacles1.txt is one of the obstacle files (you can run it with any of the provided obstacle files). You can use the graphical interface to setup the initial robot center and radius, goal center and radius, and sensor range (right click to get menu; press F1 to get help). Robot is shown in the red circle; goal region is shown in green; obstacles are shown in blue. Note that the robot does not do anything. It's your assignment to implement the different control strategies and bug algorithms.

Support code written by Erion Plaku.