Coding Challenge: Discrete planner for motion planning

Title: Discrete planner for motion planning

Description: 2D discrete path planners are designed for the navigation of a holonomic robot in a known static and flat environment. The planner could assist the robot in maneuvering though warehouse for autonomous surveillance. Two discrete planners are designed, random and optimal planner. Both the planners have a common interface, and implement the search method with the following signature:

search(world_state, robot_pose, goal_pose) return path

Inputs:

world_state is a 2D-grid representation of the environment where the value 0 indicates a navigable space and the value 1 indicates an occupied/obstacle space.

robot_pose is a tuple of two indices (x, y) which represent the current pose of the robot in *world_state*.

goal_pose is a tuple of two indices (x, y) which represent the goal in *world_state* coordinate system.

Output:

path is a list of tuple (x, y) representing a path from the *robot_pose* to the *goal_pose* in *world_state* or None if no path has been found.

Outline: Solo Iterative Process (SIP) is used to meet the obligations and manage the resources. An initial product backlog is created and maintained for the SIP Enactment spanning two weeks containing tasks and comments. Further, to ensure high quality software engineering practices, a time log containing the estimated effort required for each task is maintained. The estimates are updated on a weekly basis, depending on the previous product backlog. A defect log is maintained to measure software quality.

Program is written in C++ using 11/14 features with CMake as the build system in Eclipse IDE. Doxygen style commenting is followed. Cpplint is used for static code analysis to identify potential source code issues that conflict with the Google C++ style guide. Further, cppcheck is used to detect various kinds of bugs in the code. Unit tests using googletest framework are written to test the methods. A private GitHub repository is available, integrated with Travis. The external open-source libraries required are math, algorithm, and gnuplot-iostream [1]. The gnuplot-iostream library is used to visually demonstrate the planner output.

Algorithm:

1. Random planner: It tries to find a path to the goal by randomly moving in the environment only in orthogonal and avoiding obstacles. If the planner can not find an acceptable solution in less than *max_step_number*, the search fails. The random planner also has short memory, so that it never visits a cell that was visited in the last *sqrt(max_step_number)* steps expect if this is only available option.
2. Optimal planner: It goes to the goal with the shortest (non-colliding) path moving only in orthogonal. The planner uses A* algorithm to determine the optimal path by minimizing a cost function and heuristic function [2]. The cost function is defined as the Euclidean distance between current pose and start pose. The heuristic function is defined as the Euclidean distance between current pose and goal pose, which is never an over estimate to the true cost to reach the goal pose, hence guaranteeing us the optimality.

Risks and Mitigation: If the user enters the start or goal node lying in the obstacle space or outside the floor plan boundaries, then the algorithm will not be able to output any trajectory. Mitigation strategy will be to verify the start and goal node before running the path planner. Optimality of the algorithm may not be guaranteed if the heuristic is not admissible. This can be avoided using an appropriate heuristic function.

Deliverables: Wankhede-Ajeet.zip file including:

1. Class diagram, activity diagram.
2. SIP Enactment documents.
3. Code for the planners.
4. Unit tests.
5. README file.
6. License file.
7. Doxygen report.
8. cppcheck and cpplint outputs in a text file.
9. CMakeLists file including external dependencies.
10. Output of visual demonstration of planners.

Timeline:

	Tasks
Week 1	Project design phase: UML diagrams, class declaration, unit test implementation, CMakeList file, License file.
Week 2	Project implementation phase: Class implementation with visual demonstration, cppcheck and cpplint outputs, README file, update UML diagrams, Doxygen report.

References:

[1] http://stahlke.org/dan/gnuplot-iostream/
[2] https://en.wikipedia.org/wiki/A*_search_algorithm