American University of Sharjah
College of Engineering
Computer Science & Engineering Dept.
P.O. Box 26666
Sharjah, UAE

Instructor: Dr. Michel Pasquier
Office: ESB-2057
Phone: 971-6-515-2883
E-mail: mpasquier@aus.edu
Semester: Spring 2024

CMP 494-10 INTELLIGENT AUTONOMOUS ROBOTICS

PROJECT #2

Due date/time: Sunday 5 May 2024, at 11 pm

Instructions: Submit via the assignment box on *iLearn,* before the deadline, a single *Zip/Rar* archive that contains your *models* and *explanations* in one *Word doc* and all your *Webots project folders.* Make sure to include the *name and AUS ID* of each teammate at the start of all your files. Include them in the archive name as well, as per this format: "CMP494-PR1-SThrun74321-OKhatib64213-DKumar83524.zip". Follow the instructions below and any other complement posted later on *iLearn.* Late submissions will be penalized as per the course policy.

Note that you are to complete this assignment *as a team of three or four students.* Furthermore, each team must *work independently* and hand in their own original answers. You are *not* allowed to discuss or *share* any solution or to *copy* from others or from any sourced material. Plagiarism and cheating will be severely penalized, starting with a zero grade for the assignment. Recall you are bound by the AUS Academic Integrity Code, which you signed when joining AUS.

Assignment: In this project, you are to program a mobile robot in simulation, so that it can plan then follow various paths in a task environment, as elaborated hereafter.

System setup: You are to use the same Webots robot simulator from Cyberbotics.com, as used in class and in previous assignments, which you should be very familiar with by now. As always, the Webots user guide and reference manual are your main resources, in addition to the textbooks, class material, and your own notes.

For each project stage hereafter, create a Webots project called "Project1", "Project2", etc. that will contain your *world* model and robot *controller/s* (and other folders as created by Webots). Note that, since projects have a lot in common, you may want to create the next one by duplicating then editing the current project.

Mobile robot: Select one of the wheeled mobile robots of (roughly) circular shape that are available in the Webots simulator, such as the *E-puck, Eliza, Firebird, Khepera, iRobot Create,* etc. Note that you may want to read through the different stage

requirements hereafter before you select (wisely) your robot, based on mobility, sensors, etc. Note also that you must use the same robot in all project stages.

The robot must be able to move freely in the environment, to any position and orientation. It must be able to reach any designated waypoint or landmark, including the goal destination, as quickly as possible (i.e., minimizing travel time) and/or as safely as possible i.e., avoiding collisions with the walls and the obstacles.

Your program ought to comprise a deliberative layer, responsible for path planning at the topological level, and a reactive layer, that perceives objects and controls the motions of the robot in the environment. The robot must therefore be equipped with sensors that allows detecting the walls and the obstacles. A standard wheeled robot is sufficient (and recommended) to move around the room and achieve the tasks.

Robot programming: Decide which programming language you will use throughout the project i.e., either Python (strongly recommended), C, or C++. (Using Java is not advisable due to various issues.) Note that, as was emphasized in class, your code should look very much the same regardless, since proper abstraction should be used and the same logic should be implemented in all cases. What matters is the design, from the software architecture and the decision logic to the use of sensors and applied control directives… not the program syntax.

While your report will explain your design, models, etc. you are strongly advised to include appropriate comments in your robot controller code as well, to help the reader/instructor better understand and appreciate your effort and implementation.

Since your program is getting larger and more complex (than in previous coding assignments), structuring it well is a strict requirement, and esp. since all code should be included in a single source file. In this case, you should organize your code in layers (i.e., deliberative, reactive, sensing and control) and define for each appropriate functions. The more structured and readable your code is, the better.

Task environment: In this project, the robot's environment is a square room that contains a number of blocks representing furniture. The environment is initially static, with only the robot moving around. However, the three workstations `A-C` may be moved to a different location, as needed. Figures 1-2 below show a 2D view of the room, with the robot at the starting location `S` and its goal location `G`.

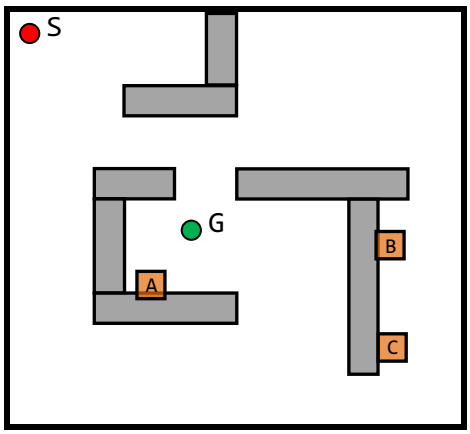

Figure 1: Task environment for the autonomous mobile robot.

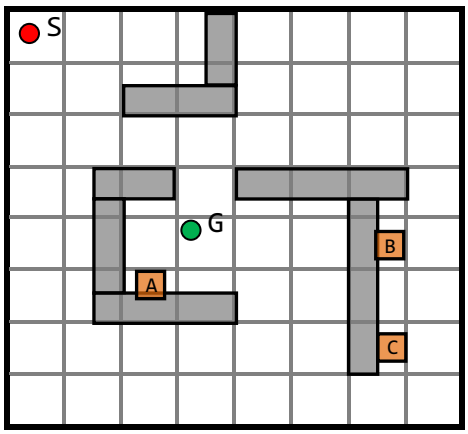

Figure 2: Task environment with a uniform (8x8) grid discretization.

In each of the following sections, you will use the same environment. You need therefore to *build a* Webots *world* that matches the 2D map shown in Figure 1. The size of the square room should be about 16 times the size of the robot (i.e., 32 times the radius, in the case of a circular robot), as depicted. This is important, to ensure that the task of moving the robot around is neither trivial nor impossible.

Each piece of furniture acts as an obstacle and can be modelled as simple block, as there is no need for details. Make sure, however, that the size and the position of each block matches closely what the map shows, including the three workstations (use a ruler to measure, if needed). Lastly, the floor should be flat and have a uniform color (i.e., do not use the chessboard-like floor, which is visually confusing).

Design and Implementation: The three stages, described in detail in the following sections, consist of designing, programming, and testing (1) the deliberative layer, (2) the reactive layer, then (3) integrating them into a functional autonomous robot.

1. Deliberative robot: Figure 3 shows another sample task environment, which is used to illustrate the steps that you need to explain and document in your report. Note, however, that your implementation will only start at the waypoint graph in Figure 6. Notice also that the configuration space will *not* used in this project (exceptionally:).

(a) *Explain* briefly but precisely why using a uniform grid map of the environment is not a good idea for path planning. Consider the given example of Figure 1 above to illustrate. *Highlight* the pros and cons of increasing the grid resolution e.g., to 16x16 and so on, and especially for a real-world environment.

(b) Instead, an adaptive grid representation of the 2D environment is created that consist of a hierarchy (a quadtree) of cells where each is either full (obstacle entirely), empty (no obstacle at all), or mixed. Any mixed cell is decomposed into four smaller cells, and the process repeats recursively, until all cells are full or empty, or the desired precision is reached. Figure 4 shows the resulting decomposition up to level 3 (where level 0 is the room, level 1 is shown in red, level 2 in green, and level 3 in blue). *Draw* a similar *quadtree representation of your environment* (i.e., for the one given in Figure 1) up to level 3 as well. *Draw* the matching *tree structure* next to it. You should label the regions on the map and nodes in the tree, for clarity.

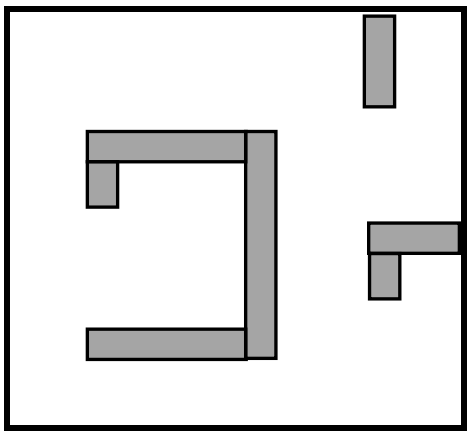

Figure 3: Sample task environment for the autonomous mobile robot.

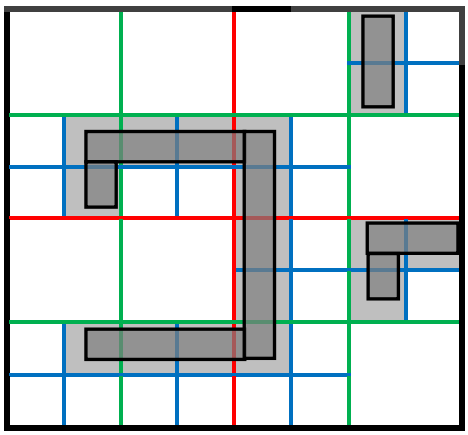

Figure 4: Equivalent quadtree decomposition (up to level 3).

(c) Next, waypoints are generated that correspond to the centers of each empty cell in the quadtree. A number of waypoints are then combined to simplify the model. Figures 5 and 6 hereafter show the waypoints before and after optimization. *Copy* the quadtree representation and matching tree structure you derived in part (b), then *add all necessary waypoints* to them. *Explain* any case where you did *combine two waypoints* into one or, if you did *not* merge any, *explain why* (using an example).

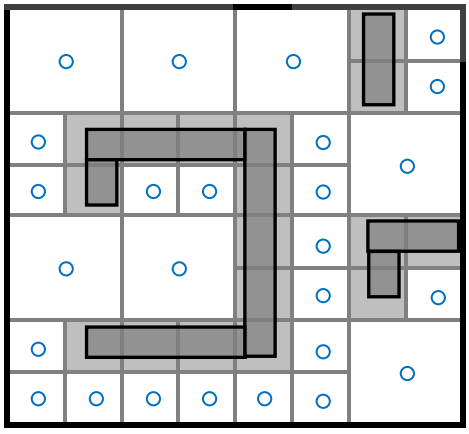

Figure 5: Waypoints generated from the sample quadtree representation.

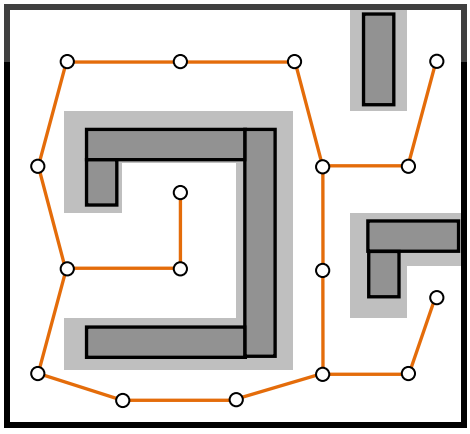

Figure 6: Topological map of the connected waypoints (optimized).

(d) *Connect your waypoints* into a *graph* data structure, as illustrated in Figure 6. Waypoints are connected based on visibility i.e., when there is no obstacle along a straight line between them. Note that the furniture will not be moved hence this graph will not change, with the exception of the two additional waypoints that are the start location `S` and the goal `G`. Add these to your graph, as per Figure 1.

Now, *define a data structure* (for your controller program) to store the waypoint graph. All waypoints are basically 2D points in the environment (where the X axis is horizontal, the Y axis vertical, and the origin at the bottom-left corner). One should be able to change start and goal easily, so as to produce different paths. Your program must therefore *define two point variables* for `S` and `G` (that we can easily edit), and *automatically add these* two to the static waypoint graph above, so that each of start and goal are always connected to the nearest waypoint, as depicted in Figure 7.

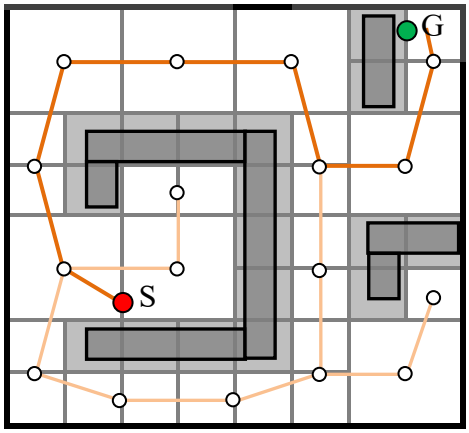

Figure 7: Example of a planned path from a start location S to a goal G.

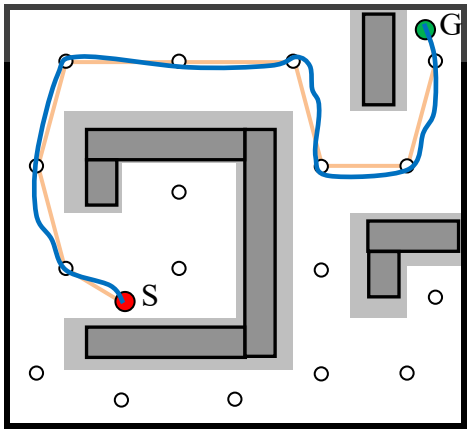

Figure 8: Example of a multi-step APF trajectory followed by the robot.

(e) *Implement the path planner* using the *A** search algorithm, to allow finding the optimal path between start and goal locations in the waypoint graph. Many implementations of *A** are available, such as from the Rosetta Code website. Use the *distance travelled* (on the graph) for the path cost and the *straight-line distance* from the robot location to the goal as the heuristic function. (If you do not use any heuristics, *A** performs like Dijkstra's algorithm – still optimal, but not efficient.) *Test* your path planner with at least *four different start-goal pairs,* the first of which should be the one shown in Figure 1. *Print out the paths* generated, including all waypoints from start to goal and their 2D coordinates on the map.

(f) *Add to the deliberative layer* i.e., the path planner developed in part (e), a function that will take the overall planned path and feed each step to the reactive controller. In the simplest case, each step should consist of a pair of consecutive waypoints in the path. Note however (and that should become clearer after testing in simulation) that it may be more efficient to provide three consecutive waypoints (not just two), on the ground that waypoints are just guides but are not destinations per se (except for the last waypoint, of course, since that is the goal). *Show the output* of this function in each of the *four test cases* you used in part (e).

Note that your program *must* be a controller in Webots (all code in one single file), even though in this stage we are not moving the robot yet. This will happen in stages 2 and 3 hereafter.

2. Reactive robot: Figure 8 above shows (in blue) the actual trajectory followed by the robot in-between waypoints using a reactive controller (e.g., APF based). It must be noted that the actual trajectory may vary from one experiment to another.

(a) *Implement the reactive layer* of your controller program so as to drive the robot from its current location or waypoint to the next given location or waypoint. You can simply reuse the Artificial Potential Field (APF) approach you implemented in Project #1, or else devise any other simple control strategy as you see fit. *Describe* concisely but clearly *your approach,* and especially the logic / strategy implemented.

Regardless of the chosen approach, the reactive controller should be able (1) to move the robot toward the next given waypoint (it is not required to pass exactly through it), also (2) to stop at (or close enough from) the goal location, and last but not least (3) to avoid any obstacle that may appear near the robot. The latter feature is needed, hence a requirement, even if the environment is static – because of control errors.

(b) *Test your controller program* in Webots, trying first to drive the robot directly from the starting location `S` to its goal location `G`, as per Figure 1. *Show a trace* of the trajectory followed by the robot and *comment on the results.* (You should of course reuse the pen device and code as per your earlier assignments.)

(c) *Test the reactivity of your controller,* trying now to drive the robot from the start location `S` straight to one near the bottom-left corner, then moving (by hand) one of the obstacles `A-C` in the path of the robot. *Show a trace* of the trajectory followed by the robot, including corrections of course, and *comment on the results.*

3. Hybrid deliberative-reactive robot: Figure 7 above depicts the nominal path generated by the deliberative layer i.e., a sequence of waypoints between start and goal, while Figure 8 shows the trajectory followed by the robot using the reactive controller.

(a) *Combine* both your *deliberative path planner* from stage 1 with your *reactive controller* from stage 2 to realize an hybrid deliberative-reactive robot architecture.

Given the environment in Figure 1 (for which you have created a waypoint graph) and user-specified start and goal locations (as per 1(d) specs), the path planner will generate a nominal path that consists of a sequence of waypoints, as can be seen in the sample world of Figure 7 again. Next, the reactive controller will direct the robot from the start location toward the first waypoint, and from there toward the second waypoint, then onto the third one, etc. until the goal destination is reached (or close enough). An example of multi-step (APF-controlled) trajectory followed by the robot in such fashion can be seen in Figure 8.

(b) *Test your hybrid controller program* in Webots, driving the robot from the starting location `S` to its goal location `G`, as per Figure 1. *Show a trace* of the trajectory followed by the robot and *comment on the results.* Repeat those steps for each of the *other three test cases* you employed in part 1(e).

(c) *Test the reactivity of your controller* again, this time repeatedly moving (by hand) one of the obstacles `A-C` in the path of the robot. That is, with reference to Figure 1, you could do it between the start and the first waypoint, then again between any two further waypoints, and finally when near the goal. *Show a trace* of the trajectory followed by the robot, including all corrections, and *comment on the results.*

Make sure your path planner prints out the sequence of all waypoints and their coordinates (as per 1(e) specs), from the start to the goal (both user-specified), before it starts moving. In addition, as the robot keeps going toward the goal, your program should *print each waypoint* as the robot passes near it. Simulation should stop when the robot reaches the last waypoint i.e., the goal destination.

(d) *Modify the path planner* to generate the *safest path* instead. Define the path cost function appropriately, and devise likewise a suitable heuristics. *Explain your design* clearly. *Test* your revised path planner with the same *four test cases* again.

Bonus: For extra credits, you may consider implementing path visualization i.e, use a Webots supervisor to draw the waypoint graph (nodes and edges) on the room floor, then draw the nominal planned path on top, using another color. Finally, your pen device will still trace the actual trajectory followed by the robot…