RI16x62: Lab 5 – Planning

Due: Tuesday, Week 8   5.0, 5.1, 5.1b
Due: Tuesday, Week 9   5.2, 5.2h

Introduction

With the conditional plan you created by hand last week, you got a taste of what planning is all about. In this lab, you will develop full-fledged planners. First you will write a sequential planner for when the robot has complete information about its position and about the environment. Next, you will have to deal with the possibility that the robot doesn't know its starting position, giving rise to sequential planning with uncertainty. Finally, you will develop a conditional planner, which is unique because the plan it generates contains branches. The common thread here is that all of these programs return plans. **NOTE: This is a two-week assignment!**

Note also that for this lab you will start showing some early results with the USB WebCam vision system. This verifies that you can make it work, a Good Thing.

For all of the assignments in this lab, we will test your code by placing the robot in a real environment, giving you the map and a problem (using a serialized file that you read directly in!) and watching your robot solve the problem, physically!

This document is not a complete spec for this lab. You will need to look at our specification of the initial conditions I, final conditions G and maze and maxdepth. Furthermore, we are providing you with an example depth first iterative deepening search system (thus almost completely doing 5.0 for you) to get you started. Go to the end of this handout for pointers to these Goodies.

Assignment 5.0: Sequential Planning with Certainty

In the easiest case, the robot has complete information about the world: it has a correct map of the environment and it knows its initial position and orientation. A sequential planner accepts a specification of this knowledge and is then capable of creating a sequential plan that, if executed, would take the robot from the initial position to the goal position.

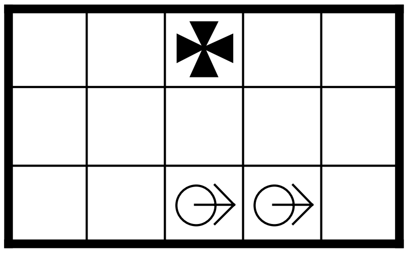
Write a sequential planner that finds and executes sequential plans when initial conditions are specified with certainty.
You may assume that the map, given to you via the initialization file, is complete and that the robot position specification will be a *singleton state set*.

For this lab, we consider the commands `gtnn(1)`, `(turn-to 900)`, `(turn-to -900) to be atomic` (i.e. the building blocks of plans). Therefore, the length of a plan is the number of atomic actions it contains.

Although a plan can only contain the atomic actions described above, you are free to optimize your plans by combining strings of `gtnn`'s into single `gtnn` calls. But remember, the `concept of maxDepth` (given in your spec file) refers to the maximum length of the unoptimized plan, not the optimized one!

Assignment 5.1: Sequential Planning with Uncertainty

Plain sequential programming was easy, but what happens in a case in which our robot has positional uncertainty?



The robot knows that it is either at `((2,0) r)` or at `((3,0) r)`. The goal is to reach node (2,2). If the robot "assumes" one of the two positions and executes a corresponding sequential plan to the goal, it will end up at node (2,2) if its assumption was correct, or node (1,2) or (3,2) if its assumption was wrong. There is a sequential solution to the problem, though:

```
GTNN(1) GTNN(1) TurnTo(900) GTNN(1) GTNN(1) TurnTo(900) GTNN(1) GTNN(1)
```

This succeeds because of the behavior of `gtnn` when the robot is facing a wall. Therefore, after the first two moves, the robot is definitely at `((4,0), r)`. This is an example of *coercion*, a movement that collapses the number of possible states.

Write an uncertain sequential planner.
When called, this planner attempts to find the shortest sequential plan no longer than `maxDepth` that would result in the robot reaching a goal state if its initial state were any state consistent with the initial conditions. If there is a solution, this function should return the solution plan. Otherwise, the function should so indicate. You may assume that the map (entered via `the initial conditions file`) is complete but the robot position specification will be a set of possible positions.
Your robot should then take the resulting plan and execute it!

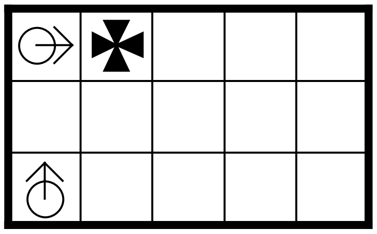
As with `Assignment 5.0`, you will need a progression function to determine the result of a applying an action to the states in a state-set. The function from assignment 4.2 does exactly this.

Assignment 5.1b: I have an Eye duh test

To literally force you to try out the USB Web camera and interface, there is an additional requirement during your sequential certain and uncertain planner execution episodes. Please have your robot take a picture at the starting location and at the ending location. The starting location picture should be displayed on screen during the run or at the beginning of the run, and when you reach the goal, switch to a picture of that. This is a forced no-brainer worth a non-zero number of points.

Assignment 5.2: Conditional Planning

You may have already noticed that sequential planning with uncertainty also has limitations. The solution plans are often extremely long and, worse, planning fails altogether in some cases. Consider the following:



There is no sequential solution to this problem. The rotational discrepancy makes the "coercion" performed in the previous example impossible. There is, however, a conditional solution:

```
GTNN(1) GTNN(1)
if (WhatDoISee() == 1100) {TurnTo(-900) GTNN(1)}
             else {TurnTo(900) TurnTo(1800) GTNN(1)}
```

Write an optimal conditional planner.
When called, this planner attempts to find the shortest conditional plan no longer than `maxDepth` that guarantees that the robot will end in a goal state if started in any state specified in the initial conditions. If there is a solution, this function should return the conditional plan and then the robot should execute it. Otherwise, the function should return `nil`. You may assume that the map (entered via `init-map`) is complete.

This is the most challenging assignment in this lab. You will certainly be using a function similar to `acts` to determine the effect of actions. Since the robot will act differently based upon the percepts it encounters, you will also need to use your `sees` function from the last lab. [Remember that `sees` takes a state-set and a percept (the result of `WhatDoISee()`) and returns the subset of that state-set that is consistent with that percept.] Your searching algorithm will be complex, branching to find the right actions as well as branching *across* plausible percepts. We will discuss this notion in more detail in class. For those of you in the know, this is AND-OR graph search. For conditional plans, the length of a plan is the maximum number of serial atomic actions (i.e. the number of actions in the longest branch of the plan).

Assignment 5.2h: Early Hints of Gold

To ensure that you start tackling issues of detecting gold, you have an additional job unrelated to the Condition Planner. We will ask you to describe to us one way you are thinking of detecting gold on the walls. And you will demonstrate for us this technique by placing and removing gold from the robot's field of view, and responding in some interesting way (examples: music, screams, screen changes, jiggles, etc.). We are not looking for bulletproof code, but for something that, under the right conditions, does work.

(Mandatory) Goodies

There are three sample project relevant to you these two weeks. First of all, check out SamplePlanner in the Samples directory. Note its use of MazeWorld. This is the class that we will use from now on to specify the problem to you. You will have to include this class of course, from now on. Now, SamplePlanner gives you a whole lot of 5.0 and 5.1 because, well, because it works.

At the same time, you need to learn how to read in MazeWorld files. Well, MazeDisplay is the second sample project for you to look at, and it does just that, It reads in MazeWorld problems and displays them using MazeGraphics. Also useful you may find MazeEditor project, which allows you to create your own problems using a GUI and then save them to a file.

When you want to load up some mazes, for fun, from MazeDisplay, then go to the Mazes directory in 16x62 and check out ?. Note that our goals are always direction-less positions from here onwards. That's why the goal is shown, just as a position, using the gold brick. Check out Maze1. For some real excitement check out Maze2. Two possible initial conditions (two states in I) and 8 possible goal states (8 states in G)- and we don't care which we end up in. Is there a sequential plan from I to G? You bet there is!